MASTER EN INGÉNIEUR DE GESTION À FINALITÉ SPÉCIALISÉE EN DATA SCIENCE

Critères techniques de succès d'une application dans l'App Store

Umuhoza, Patience

Award date:
2020

Awarding institution:
Universite de Namur

Critères techniques de succès d'une application dans l'App Store

Patience UMUHOZA

Directeur: Prof. C. BURNAY

Mémoire présenté
en vue de l'obtention du titre de
Master 120 en ingénieur de gestion, à finalité spécialisée
en data science

ANNEE ACADEMIQUE 2019-2020

Avant-propos

Je tiens à remercier les différentes personnes sans qui la réalisation de ce mémoire n'aurait pas été possible.

Je remercie tout particulièrement le directeur de ce mémoire, M. Corentin BURNAY pour sa disponibilité, ses précieux conseils, son temps et sa relecture qui m'ont permis de délivrer ce travail.

Je souhaite également remercier de manière plus générale, l'ensemble des enseignants dont j'ai pu apprendre le meilleur durant mes années d'études.

Pour terminer, je remercie bien évidemment mes parents, Benie MUREKATETE et Martin DUSABE, et mes amis pour leur soutien et leurs encouragements durant la réalisation de ce mémoire.

Table des matières

1 Introduction

1.1 Contexte

La récente croissance des technologies de l'information issue de la révolution numérique a pris d'assaut le monde et a favorisé le développement des smartphones et de leurs systèmes d'exploitation, ce qui a conduit à l'augmentation de l'utilisation des applications mobiles [10]. En effet, l'industrie des Technologies de l'Information et de la Communication (TIC) a atteint un niveau supérieur et ce, grâce à l'émergence de nouveaux appareils mobiles et à la croissance impressionnante des différents services d'applications mobiles offerts dans les magasins d'applications [12].

Une application est un logiciel que toute personne en possession d'une plateforme appropriée peut installer sans nécessité d'une expertise technique [14]. On la qualifie de "mobile" lorsqu'elle est exécutée sur des appareils mobiles, tels que les smartphones ou les tablettes, et qu'elle exécute certaines tâches spécifiques pour son utilisateur [15]. À l'heure actuelle, les consommateurs désirent avoir accès à un maximum de services à tout moment et à tout endroit, accroissant ainsi l'attrait pour les smartphones et leurs applications mobiles. Pour répondre à la demande grandissante, de nouveaux types d'applications ont ainsi vu le jour dans une multitude de domaines différents tels que la santé, le fitness, le divertissement ou encore le shopping.

L'une des raisons majeures de l'explosion du secteur des applications mobiles est le fait que ce secteur est particulièrement lucratif pour les acteurs tels que les vendeurs d'appareils, les détenteurs de plateformes, les fournisseurs de services, etc [12]. En effet, la plupart des individus vivant dans des pays développés ne quittent plus leur domicile sans leur téléphone mobile [9] qui ne leur sert plus uniquement de moyen de communication mais constitue un véritable outil de la vie quotidienne duquel ils ne peuvent se séparer. L'évolution des smartphones, la croissance de leur utilisation et le développement des données internet mobiles (i.e. données cellulaires) ont eu pour conséquence d'accroître la dépendance des consommateurs à leurs téléphones mobiles. Car, en effet, ces facteurs ont permis aux utilisateurs de faire toute une série de choses – qu'ils faisaient jusqu'alors depuis un bureau – à distance par le biais de leur téléphone mobile [9]. Et ça, les entrepreneurs de tout type, l'ont bien compris. C'est pourquoi il est devenu nécessaire pour ces derniers de développer des applications capables de répondre au mieux aux besoins des consommateurs voire même de les anticiper. L'aspect quantitatif en termes du nombre d'applications développées en vue de répondre aux besoins des consommateurs n'est dès lors pas un problème. Mais encore faut-il qu'elles soient aptes à répondre aux attentes des utilisateurs pour pouvoir survivre dans un marché aussi concurrentiel que celui des applications mobiles.

1.2 Problème

L'industrie des applications mobiles repose sur les magasins d'applications qui constituent des plateformes sur lesquelles ces applications mobiles sont distribuées [15]. Sur base du type de smartphone employé, les utilisateurs se rendent sur le magasin d'applications correspondant au système d'exploitation utilisé par leur smartphone afin d'accéder aux applications mobiles dont ils peuvent disposer. Il existe donc plusieurs magasins de ce type, chacun dépendant du système employé par le smartphone. On distingue deux grands leaders sur le marché : l'App Store d'Apple et le Play Store d'Android [7], tous deux lancés en 2008. Il en existe d'autres, de moins grande envergure, comme le BlackBerry World de Blackberry ou encore le Windows Store de Windows [14]. Ce mémoire se focalisera sur l'analyse de l'AppStore, doté d'une API sur laquelle sont publiées des données techniques relatives à chacune des applications.

La littérature actuelle offre une multitude d'analyses et de conseils sur le développement des applications, sur la façon de choisir le marché sur lequel les publier ou encore sur les feed-backs des utilisateurs. Ces analyses de feed-back sont généralement réalisées en vue d'expliquer les raisons des échecs commerciaux de certaines applications mobiles. Compte-tenu du fait que les utilisateurs sont ceux qui décident de télécharger, d'utiliser, de noter et de commenter les applications, c'est donc eux qui définissent le succès ou l'échec d'une application mobile. En l'occurrence, la note (i.e. le rating) attribuée aux applications est constituée par la moyenne des ratings accordés par les utilisateurs à l'aide d'une évaluation allant de 1 à 5 étoile(s) [15] ; 1 étoile correspondant à une expérience désastreuse et 5 étoiles à une expérience exceptionnelle. Selon [15], le rating d'une application est un indicateur important de la satisfaction perçue par les utilisateurs, ce qui fait sens compte tenu du fait que d'après [16], les ratings reflètent généralement l'expérience des utilisateurs lors de l'utilisation de l'application. Dès lors, le succès d'une application est ici défini par son rating correspondant. Compte-tenu de tous ces éléments, il semble dès lors intéressant d'analyser l'aspect technique des informations disponibles des applications en vue de détecter s'il existe un lien de causalité entre les attributs techniques des applications publiées et leur succès. Cette analyse vise dès lors à émettre des recommandations aux managers d'entreprises désireuses de lancer leur propre application. La question de recherche de ce mémoire est donc la suivante :

« Quels sont les critères techniques de succès d'une application dans l'App Store ? »

1.3 Approche utilisée

Afin de tenter de répondre au mieux à cette question de recherche, il est primordial de réaliser une revue de la littérature existante. Cette revue est disponible dans la deuxième section de ce mémoire et reprend une description des applications mobiles et de l'App Store, une analyse des feed-backs des utilisateurs ainsi que les facteurs à prendre en compte lors du développement d'une

application. Cette section sera suivie de la troisième section, contenant une base méthodologique rassemblant quelques méthodes de prédiction de classification. Celle-ci sera également suivie d'une quatrième section qui reprend les mécanismes utilisés pour capter les données issues de l'API de l'App Store, pour mettre en place l'ensemble de données ainsi que pour traiter et analyser ces dernières. Une cinquième section fournit, ensuite, une description de l'ensemble de données construit et une analyse des résultats de la recherche. Ensuite, dans une sixième section, est présentée une discussion des résultats et de ce qu'ils impliquent pour un créateur d'application sur l'App Store. Enfin, dans une septième section sont développées les explications des limites rencontrées. L'ensemble permet de dégager, dans la huitième et ultime section, les conclusions sur la question de recherche.

2 Revue de la littérature

2.1 Applications mobiles

Le secteur des applications mobiles est un nouveau secteur en croissance qui constitue un segment des TIC qui se développe rapidement [9]. En effet – et c'est ce qui le différencie principalement du secteur des services traditionnels – ce secteur fournit ses services aux consommateurs peu importe l'heure et l'endroit [12] et c'est un service dont la consommation se fait en continu [19]. De plus, les applications mobiles représentent une source de revenu capitale pour l'industrie des téléphones mobiles et des logiciels [12] ; c'est pourquoi les entreprises ont infiltré ce marché en développant des services de toute sorte pour leurs clients.

Durant ces dernières années, une innovation à grande échelle a eu lieu dans différents secteurs de services mobiles tels que les services de contenu ou encore les services traditionnels hors-ligne ce qui a donné lieu à un grand nombre de nouvelles catégories dans les magasins d'application [12]. On peut regrouper lesdits services de la manière suivante, proposée par [9] :

— Les applications centrées sur la communication qui regroupent, entre autres, les navigateurs internet (e.g. google chrome, safari, mozilla,...), les plateformes e-mail (e.g. gmail, hotmail, yahoo,...) ou encore les réseaux sociaux (e.g. facebook, Instagram, Twitter) ;

— Les applications de jeux en tous genres, que ce soit des puzzles (e.g. Candy Crush, Block !, 2048,...), des jeux d'aventure (e.g. les Sims, Warface, Criminal Case,...), de mémoire (e.g. Peak, Elevate,...) ou encore de stratégie (e.g. Gladiator Heroes, Battle Islands, Narcos,...) ;

— Les applications de multimédia telles que les lecteurs audio (e.g. Imusic, Musique, Spotify,...), les lecteurs vidéo (e.g. VLC, MX Player, PlayerXtreme,...) ou encore les visionneurs d'image (e.g. Photos, Google Photos,...) ;

— Les applications de productivité telles que les calendriers, calculettes, les notes, la suite Office ou une autre suite bureautique, et bien d'autres ;

— Les applications utiles de voyage telles que les convertisseurs de devises, les traducteurs, les applications GPS, les applications météo et autres ;

— Les applications utilitaires telles que le dictaphone, le gestionnaire de profil, le carnet d'adresse, le gestionnaire de tâches, d'appel, de fichiers et autres ;

Une segmentation peut également être faite sur base de la valeur inhérente du service mobile qui est créée par l'utilisation des services de l'application mobile en termes de consommation et d'expérience client. Cette valeur peut être utilitaire et comprend donc les applications mobiles qui aident les utilisateurs à accomplir leurs obectifs de manière efficace, ou hédonistique et englobe donc les applications qui créent des expériences fun pour les utilisateurs et sont uniquement utilisées à cette fin [12].

De manière générale, les applications mobiles peuvent être gratuites (et généralement financées par les publicités), payantes ou fonctionner « à l'essai » gratuitement pendant une période déterminée avant de devoir souscrire un abonnement premium [3]. Cependant, la distribution des types d'applications en termes de prix ne se fait pas de manière proportionnelle. En effet, selon l'étude de [12], 30% des applications de son ensemble de données, toutes catégories confondues, sont gratuites et plus de 50% des applications payantes sont peu coûteuses. De plus, son étude a également révélé que les 80% des applications mobiles founissant des informations simples ont des prix peu élévés, voir sont gratuites. En revanche, les applications mobiles qui fournissent des informations professionnelles sont, de leur côté, plus chères et peuvent même s'avérer assez onéreuses dans certains cas.

Les applications mobiles sont dotées d'un certain nombre d'attributs qui peuvent être de différents types. Ils peuvent, tout d'abord, être techniques et, dans ce cas, être directement issus de l'application mobile. Ils peuvent également être non-techniques et être strictement fournis par le magasin d'application sur lequel ils sont publiés. Ou bien, ils peuvent faire partie des deux catégories à la fois [14]. Ces informations sont généralement agrégées et publiées sur l'API du magasin d'application sur lequel elles sont publiées. Elles peuvent être publiées dans 1 ou plusieurs magasin(s) d'application [3]. Ce choix d'implantation se fait sur base de différents facteurs, qui sont discutés dans le point suivant, tout en gardant à l'esprit qu'il y a peu de différences entre les attentes des clients des différentes plateformes. Par exemple, pour iOS et Android, une application qui est fortement demandée sur une plateforme l'est aussi sur l'autre [4].

2.2 Plateforme App Store

L'App Store est un magasin d'applications virtuel développé et entretenu par Apple qui constitue une plateforme de distribution d'applications digitales pour iOS et qui est développé et entretenu par Apple [12]. Il fonctionne comme un marché ouvert où les utilisateurs peuvent parcourir et télécharger des applications mobiles de l'iTunes Store. L'écosystème d'Apple est un ecosystème est divisé en plusieurs sous-écosystèmes qui sont chacuns dotés d'un magasin d'applications qui fournit des logiciels pour leurs appareils correspondants (e.g. iPhone, iPad,...) [8].

Avant que les magasins d'application ne voient le jour, les applications mobiles étaient distribuées sur des portails multi-modaux opérés par des vendeurs, des opérateurs de réseaux mobiles, des fournisseurs de contenu mobile indépendants et des éditeurs d'appplications et de marques [12]. Les magasins d'applications se sont différenciés en apportant une plateforme facile d'emploi avec de la disponibilité, de la compatibilité entre les applications et les appareils, de la variété dans l'offre de produits et des contenus publiés par les utilisateurs. Ces derniers représentent la plus-value principale apportée par les magasins d'application. En effet, ils founissent un service centralisé rassemblant un grand nombre d'applications avec leurs informations techniques, leurs feed-backs provenants des utilisateurs et des données de performance commerciale relatives à

ces applications [14]. La performance de ces applications mobiles peut être mesurée par le biais du listing des téléchargement, des ratings et des commentaires laissés par les utilisateurs [13]. La particularité de l'App Store est qu'il offre un concept d'ouverture qui simplifie fortement le développement d'applications mobiles car il permet à des développeurs avec différents niveaux de compétences de créer des applications mobiles [12]. Ce principe a créé un nouveau paysage de compétitivité dans l'industrie mobile en stimulant le développement de nouvelles applications et l'innovation dans les applications offertes, d'où la création de nouvelles catégories dans l'API de l'App Store.

L'App Store suit la théorie des marchés bilatéraux où l'orchestrateur de l'écosystème, c'est-à-dire Apple, pousse les vendeurs de contenu et les consommateurs de ce contenu à s'engager dans l'écosystème mobile pour mener à une boucle de rétroaction positive [7]. Dans ce type de marché, il existe des effets de réseaux inter-groupes ce qui sont intéressants pour tout éditeur cherchant à savoir sur quel marché publier son application mobile. Cependant, l'App Store n'est pas le seul à suivre la théorie des marchés bilatéraux. C'est pourquoi il est important de décider dans quel(s) écosystème(s) publier son application mobile. Il existe un grand nombre d'éditeurs qui ne créent qu'une seule application, et ce, à des fins purement divertissantes, sans chercher à toucher un grand public [8]. Cependant, il existe aussi des éditeurs qui préfèrent opter pour l'hébergement de leur application mobile sur plusieurs écosystèmes ; que ce soit au sein d'un même écosystème [8], ou sur différentes plateformes [7]. La stratégie de multi-hébergement est généralement utilisée pour gagner en téléchargements étant donné que le type et la popularité des applications qui adoptent cette stratégie ne diffèrent pas significativement de celles qui se contentent de se positionner sur une seule plateforme d'hébergement.
La décision d'adopter ou non cette stratégie repose donc sur une mise en balance de la part de marché et du coût de l'implantation. Il s'agit, en d'autres termes, pour le créateur de se demander s'il est possible d'acquérir une part de marché qui compense le coût de l'implantation sur différentes plateformes. Selon [7], cette stratégie est principalement adoptée par une minorité d'éditeurs et d'après [8], il y a un lien de corrélation significatif entre les la taille de l'éditeur et sa propension à appliquer cette stratégie. De plus, une application choisit de s'implanter sur une plateforme seulement dès lors que la demande sur cette plateforme est assez élevée pour couvrir les coûts occasionnés par cette implantation en termes d'implémentation et de marketing [4]. Dès lors, ces facteurs impliquent donc que le coût d'implantation est souvent le facteur qui entrave la mise en place du multi-hébergement. Par exemple, d'après [4], les entreprises cotées en bourse évitent les coûts occasionnés par la stratégie de multi-hébergement car elles ont déjà une relation établie avec leurs clients et n'en éprouvent dès-lors pas le besoin. C'est pourquoi le fait que l'App Store soit une plateforme ouverte avec une Interface de Programmation d'Application (API) ouverte [12] rend celle-ci plus attractive pour les développeurs ne disposant pas des ressources nécessaires pour s'implanter sur plusieurs écosystèmes.

2.3 Critères qualitatifs de succès d'une application mobile

Dans la littérature existante, les critères qualitatifs de succès ont principalement été évalués sur base des feed-backs laissés par les utilisateurs sur le magasin d'application sur lequel les applications mobiles sont publiées. Les feed-backs sont constitués par le rating laissé par l'utilisateur ainsi que le commentaire éventuel qui l'accompagne. Ces ratings permettent fournir une évaluation de qualité, orientée utilisateurs, des applications mobiles ainsi que des recommandations pour les utilisateurs futurs potentiels [16].

De manière générale, la majorité des feed-backs qui ont été recensés sont relatifs aux applications mobiles gratuites car la communauté utilisatrice de ce type d'application est plus importante. Cependant, la longueur des feed-backs augmente avec le prix des applications, même si aucune corrélation linéaire n'a été trouvée entre les 2 [16]. Il a également été relevé que seule une communauté réduite d'utilisateurs est très active et donne régulièrement des feed-backs sur les applications. Dès lors, lorsqu'une personne ne faisant pas partie de cette communauté prend le temps de laisser un commentaire, c'est qu'elle a un vrai message à faire passer. On remarque également que moins les utilisateurs ont aimé une application plus leur commentaire est long. Ils y développent généralement leur expérience personnelle pour justifier le rating qu'ils donnent et proposent des améliorations.

Les critères recensés sur base de ces feed-backs sont essentiellement des conseils pour attirer l'attention des consommateurs et la garder par la suite. Ces critères peuvent être regroupés en différentes catégories :

2.3.1 Remarquabilité

Le marché des applications mobiles est un marché très compétitif. Dès lors, pour qu'une application reçoive un bon rating, elle doit, au préalable, faire en sorte d'être remarquée par ses utilisateurs potentiels. S'il existe un aspect sur lequel les plateformes de distribution sont semblables, c'est sur la manière dont elles connectent les applications aux utilisateurs ; ce mécanisme est appelé les "top Lists" [4]. Ces listes utilisent des méthodes de filtrage collaboratives pour tenter d'identifier les applications les plus utiles ou les plus populaires et les listent.Compte-tenu du fait que les magasins d'application et ces mécanismes sont les seuls moyens par lesquels les applications parviennent aux utilisateurs, les créateurs de ces applications mobiles sont contraints de faire en sorte de se retrouver dans ces "top lists" en usant de méthodes pour tromper le système et pouvoir gagner en visibilité. L'utilisation du processus d'App Store Optimization (ASO) est une bonne solution pour y parvenir [3]. Ce dernier préconise l'utilisation de mots-clés ou de phrases spécifiques qui permettent aux utilisateurs d'arriver plus facilement à trouver l'application lorsqu'une recherche est effectuée sur des modules de recherche.

Une fois que l'application arrive sur le moteur de recherche de l'utilisateur, c'est là que tout le processus d'acquisition commence [19]. Lorsque l'utilisateur se trouve face à une liste d'applications, il décide d'en choisir une en appuyant

sur l'écran pour afficher ses informations détaillées, comme sa description, ses captures d'écran ou encore ses commentaires. Si l'utilisateur estime qu'il apprécie ce qu'il voit et est intéressé par ses informations, il peut décider de la télécharger. Sur base de ça, il peut la tester pour voir s'il souhaite la garder ou non. En revanche, s'il n'apprécie pas ces informations à première vue, il peut simplement revenir à la liste des applications et continuer à chercher une application qui correspond mieux à ses attentes. L'intérêt pour l'application qui pousse l'utilisateur explorer ses informations se base donc sur l'attrait de celle-ci. C'est pourquoi il est crucial que celle-ci soit dotée d'un logo qui soit attractif au premier abord. En effet, un logo attractif pousse l'utilisateur à vouloir aller chercher plus loin. S'agissant du premier élément porté au regard de l'utilisateur, il faut qu'il soit significatif tant dans le choix des couleurs que de l'icône.

La remarquabilité est le critère capital de succès d'une application car c'est par là que tout commence. Dès lors, il est crucial de faire en sorte que l'application sorte du lot, non seulement de manière tactique mais aussi de manière estéthique.

2.3.2 Esthétique de l'application

Une fois l'application remarquée et téléchargée, il faut retenir les utilisateurs car il y a beaucoup de tentation sur le marché des applications mobiles [19]. De manière générale, les utilisateurs sont sensibles à l'esthétique du contenu en face duquel ils se retrouvent c'est pourquoi le design de l'Interface Utilisateur (UI) est la raison première de la désinstallation d'application mobiles [3]. Dès lors, fournir des interfaces d'applications attractives au regard implique que celles-ci s'adaptent au format de l'écrant sur lequel elles sont exécutées. Il faut dès lors prioritariser l'utilisation d'un design réactif dans la conception de l'application. D'un point de vue visuel, l'application doit être compatible avec un écran tactile car les appareils mobiles sont principalement de ce type. Elle doit également avoir l'air amusante et intuitive, être plaisante à utiliser, colorée, lumineuse, pourvue d'effets ou de belles images. Cependant, il faut faire attention à ne pas trop charger les interfaces pour que le temps de charge ne soit pas trop élevé. Une application qui est trop lente entrave les performances de l'application et a donc plus de chances de facilement se faire remplacer par une autre et aboutir à un mauvais rating.
L'aspect non-attractif du contenu d'une application mobile est un facteur majeur de perte des consommateurs toutes catégories d'application confondue [5], c'est pourquoi il est primordial d'y accorder de l'importance lors de la réalisation de l'application.

2.3.3 Caractéristiques de l'application

Dans l'absolu, pour que l'application soit attractive sans équivoque, elle devrait régler, de manière mobile, un problème existant [3], ne pas avoir d'alternative et avoir le monopole sur le marché. Malheureusement, comme susmentionné, le marché des applications mobiles est un marché en essor très compétitif, il y

a donc très peu de chances que ça arrive. Dès lors, elle doit être dotée de caractéristiques spécifiques qui lui évitent de perdre ses utilisateurs.

L'une des caractéristiques principales que doit avoir l'application est le fait qu'il doit y avoir une certaine consistance entre le prix de l'application et ses performances. En effet, le prix d'une application mobile est l'un des sujets de plainte les plus fréquents dans les commentaires des utilisateurs [5]. Développer une application mobile qui fournit un mauvais rapport qualité-prix est dangereux pour les ratings de l'application. Et même lorsque l'application n'est pas mauvaise, dès lors qu'un utilisateur décide d'investir dans une application, il s'attend à un produit qui vaut le coût d'être payé. Compte-tenu du fait que le marché des applications mobiles fournit un nombre très important d'applications issues de différentes catégories, il est très probablement facile de trouver une alternative à l'application payante à un coût moindre ou gratuite. De plus, pour ce qui est des applications autres que les jeux, les utilisateurs sont plus difficiles en termes de satisfaction de leurs attentes [5]. Autrement dit, les consommateurs préfèrent payer pour des jeux de haute qualité plutôt que pour des applications mobiles de haute qualité, donc s'ils décident d'investir, il faut que ça en vaille la peine.

L'application doit fonctionner hors-ligne pour un minimum de ses fonctionnalités. Dans le cas contraire, elle n'offrirait rien de plus qu'un site web à l'utilisateur qui ne trouverait aucun intérêt à la garder [3]. De plus, si elle a un site web correspondant, elle doit prendre en compte le fait que, sur base de la plateforme sur laquelle est distribuée, elle ne doit pas simplement être un modèle réduit du site web d'où elle est issue mais bien prendre au compte le public qu'elle vise et s'adapter à celui-ci.

L'application doit faire l'objet de mise à jour régulières en vue de régler les problèmes éventuels en portant une attention toute particulière à ceux qui sont soulevés par les utilisateurs dans les commentaires car il lui faut veiller à conserver l'attrait de ces derniers pour son application. Cependant, ces mise à jour doivent être bénéfiques pour les utilisateurs. En effet, certaines mise à jour sont faites en vue d'effectuer des améliorations ou d'ajouter des nouvelles caractéristiques à l'application. En revanche d'autres sont uniquement faites en vue d'actualiser les bibliothèques de publicité publicités. De plus, sur base des commentaires, apporter des modifications à une application est parfois source d'insatisfaction des utilisateurs car ils n'apprécient pas que des caractéristiques de leur applications disparaissent ou soient modifiées [16]. Cependant, d'après [13], les applications populaires ont tendance à avoir une fréquence de mise à jour élevée. Ainsi, même si elles ne sont pas bénéfiques pour les utilisateurs, cela ne porte pas préjudice à la popularité de l'application.

Selon [10], la description de l'application est un facteur important dans le choix de l'utilisateur de la télécharger ou non. Les critères impactant le rating sont les suivants : la mention des permissions relatives aux données personnelles, le nombre de paragraphes ainsi que le nombre moyen de mots par caractéristique de l'application.

2.3.4 Performance

La performance est également l'un des aspects les plus importants dans la rétention d'utilisateurs. En effet, une application qui crash fréquemment ou qui ne donne pas les résultats attendus est plus propice à faire l'objet d'un mauvais rating [5]. Selon [3], c'est la deuxième raison de désinstallation d'applications mobiles. Le but de toute application est de s'exécuter sans aucun bug. Comme c'est rarement possible, il s'agit alors de minimiser les bugs pour maximiser la performance.

Lors de la conception des interfaces, il faut faut attention de ne pas les surcharger car la surcharge du cache a tendance à ralentir le système et, par conséquent, à perturber la performance, impactant ainsi négativement la participation des utilisateurs.

Les applications fonctionnent sur batteries et donc la consomment. Une consommation trop importante peut mener l'utilisateur à désinstaller l'application, lui préférant une alternative moins gourmande en énergie.

3 Bases méthodologiques

Lorsqu'on tente de prédire la classification de données dans certaines catégories spécifiques, l'utilisation du machine learning se présente comme étant un bon moyen d'y parvenir. Pour se faire, il faut passer par le traitement des caractéristiques pour ensuite sélectionner un modèle de classification.

3.1 Traitement des caractéristiques

Les données d'un ensemble de données sont constituées par des instances qui sont elles-même décrites par des caractéristiques. Le problème qui se pose lorsqu'on souhaite réaliser des prédictions est que ce nombre de caractéristiques peut devenir très élevé dans certains cas. Un ensemble de données qui fait des prédictions avec trop de caractéristiques constitue un processus coûteux et a tendance à donner lieu au phénomène de surapprentissage (overfitting) [6]. Lorsque ce phénomène a lieu, il donne également lieu à de mauvaises prédictions et par conséquent, à des erreurs de classification [6]. En effet, les données de haute dimensions sont souvent composées de données dotées de caractéristiques qui peuvent être non-pertinentes, trompeuses ou redondantes ce qui augmente la taille de l'espace de recherche et qui mène à un traitement de données plus laborieux pour les modèles de classification [11]. Pour pallier à ce problème, la nécessité de procéder à une réduction de dimensions sest imposée comme une exigence pour pouvoir réaliser des classifications plus exactes sur des ensembles de données à taille importante [6]. La réduction de dimensionnalité est un outil de prétraitement de données qui est utilisé pour l'analyse, la visualisation et la modélisation de données de haute dimensions [11]. Les 2 approches classiques pour ce faire sont la sélection de caractéristiques et l'extraction de caractéristiques.

3.1.1 Sélection de caractéristiques

La sélection de caractéristiques est le moyen le plus simple de procéder à une réduction de dimensionnalité [6]. Ce processus fonctionne en sélectionnant un sous-ensemble de caractéristiques qui est uniquement composé de dimensions des données initiales contenant des informations pertinentes pour résoudre le problème. Sur base de ça, le sous-ensemble sélectionné durant le processus est celui qui contient le moins de caractéristiques mais qui celles-ci sont celles qui contribuent le plus à une prédiction exacte. L'avantage de ce processus est que les informations importantes contenues par les caractéristiques sélectionnées ne se perdent pas. Cependant, si un petit ensemble est selectionné, les informations importantes des autres caractéristiques sont, elles, perdues [11].

Les méthodes de sélection de caractéristiques peuvent être classées en 3 catégories :

— Filters : processus qui extrait les caractéristiques sans qu'il n'y ait aucun apprentissage qui soit inclus. Il peut utiliser une méthode univariée qui

considère chaque caractéristique séparément ou une méthode multivariée qui parvient à trouver des relations entre les différentes caractéristiques [6]. Le fait que ce soit un processus qui ne prend pas en considération le classificateur utilisé a pour avantage de le rendre plus efficace en termes de coût de calcul. Cependant, malgré le fait que ce soit un processus qui soit assez adapté pour les ensembles de données importants, il n'est pas très digne de confiance pour les classifications.
Un exemple de méthode Filter est le Gain d'Information qui mesure l'accroissement de l'entropie entre lorsqu'une caractéristique est présente et lorsque ce n'est plus le cas [11] ;

— Wrappers : processus qui utilise des techniques d'apprentissage pour detecter quelles caractéristiques sont utiles [6]. Il a tendance à donner de bons résultats car le processus de selection est optimisé pour le classifieur en question. Cet avantage mène directement à son inconvénient principal qui est le fait que c'est une méthode assez coûteuse car son coût de calcul devient élevé dès lors que la dimension des ensembles de données s'accroit [11].
Un exemple de méthode Wrapper est l'Algorithme Génétique qui fait partie des algorithmes évolutionnaires. Il trouve le plus petit ensemble de données pour lequel le critère à optimiser ne se déteriore pas [6] ;

— Hybrids : processus qui combine la sélection de caractéristiques et le classificateur [6]. C'est une technique qui donne généralement des meilleurs résultats que les Wrappers mais, étant donné que la sélection est influencée par les hypothèses que le classificateur fait, le sélection effectuée pourrait ne pas être l'ensemble optimal avec d'autres classificateurs.
Un exemple de méthode Hybride est la Forêt Aléatoire qui crée un certain nombre d'arbres de décision en ré-échantillonnant l'ensemble de données initial de manière aléatoire et qui utilise un système de vote pour procéder à la classification. En plus de ça, cette méthode fournit également l'importance relative des différentes caractéristiques durant le processus de classification [17] ;

3.1.2 Extraction de caractéristiques

L'extraction de caractéristiques est une méthode plus générale de réduction de dimensionnalité. Ce processus fonctionne en développant une transformation de l'espace initial en un sous-espace à dimension plus réduite tout en préservant la plupart des informations pertinentes. Dè lors, la taille de l'espace de caractéristiques est réduit sans vraiment perdre d'informations issues de l'espace initial. Le problème principal de cette méthode est le fait qu'elle ne fournit pas réellement d'information sur la contribution des caractéristiques initiales et par conséquent, les résultats sont difficilement interprétables dès lors qu'on souhaite déterminer le poids des facteurs menant à la classification obtenue [11].

Les méthodes d'extraction de caractéristiques réduisent la complexité, fournissent une représentation simple des données et sont connues pour être des méthodes moins coûteuses que les Wrappers [6]. La méthode la plus connue

est l'Analyse en Composantes Principale (ACP) qui est une méthode non-paramétrique utilisée pour extraire les informations les plus pertinentes d'un ensemble de données redondantes ou contenant du bruit [11]. Le bruit dans un ensemble de données peut être défini comme une erreur dans la variance d'une caractéristique [6]. Les algorithmes de machine learning sont souvent affectés par le bruit, alors que celui-ci doit être réduit le plus possible pour éviter de générer de la complexité inutile et pour améliorer l'efficacité des algorithmes. L'ACP parvient donc à gérer le bruit tout en exécutant une transformation linéaire des données en minimisant la redondance grâce à la covariance et en maximisant l'information par le biais de la variance [11]. Il existe également une ACP non-linéaire qui effectue également des transformations des caractéristiques initiales pour en générer d'autres qui sont plus significatives. La différence entre celle-ci et l'ACP linéaire est que cette méthode est adaptée à des caractéristiques qui se mesurent différemment.

3.2 Sélection du modèle de classification

Il existe un grand nombre de méthodes de classification disponibles ; que ce soit en apprentissage supervisé (e.g. K Plus Proches Voisins, Machines à Vecteurs de Support ou encore Arbres de Décision) ou en apprentissage non-supervisé (e.g. K-Moyennes ou encore le Regroupement Hiérarchique). Une fois le traitement des caractéristiques effectué, il faut faire passer les données dans un algorithme de classification. Si le mode de traitement utilisé incluait l'utilisation d'un classificateur comme la Forêt Aléatoire par exemple, il n'est pas obligatoire de faire passer les données dans un nouvel algorithme mais tout dépend des attentes de l'utilisateur.

Compte-tenu du choix offert, la méthode à utiliser pour choisir le modèle le plus optimal peut se faire par le biais d'un algorithme de sélection de modèle [18]. Sur base des attentes de l'utilisateur et des données disponibles, certains modèles seront plus appropriés pour effectuer certaines tâches. Par exemple, une Forêt Aléatoire ne nécessite la définition que de 2 paramètres obligatoires ou encore, les Machines à Vecteurs de Support sont plus appropriés pour les problèmes binaires. C'est pourquoi, sur base d'un ensemble de méthodes pré-selectionnées, l'utilisation d'un GridSearchCV sur ces méthodes permet de retourner un modèle optimal avec les paramètres correspondants.

4 Méthodologie

Une méthode de recherche exploratoire a été menée sur plusieurs étapes en vue de déterminer les critères techniques de succès d'une application dans l'AppStore. Cette méthode démarre avec la récupération des données techniques relatives à différentes applications publiées sur le marché d'applications qu'est l'AppStore d'Apple. Ces données sont récupérées via l'API de l'iTunes Store d'Apple [1]. L'étape suivante est le traitement des données en procédant, dans un premier temps, à un prétraitement des données en vue d'éliminer les variables qui n'apporteront rien ou peu à l'analyse et de formater les données restantes. Dans un second temps, un traitement des données sera effectué par le biais d'un modèle de classification supervisé.

4.1 Recueil des données

Premièrement, les données contenant les informations techniques des applications ont été récupérées sur l'API de l'iTunes Store. Ces données ont été collectées par le biais d'un script écrit dans le langage de programmation Python. Ce script, qui est disponible à l'annexe B, utilise les modules et packages suivants :

- **Urllib3** : Module client de http permettant de rassembler les connexions, d'aider à relancer des requêtes ou encore d'aider à gérer les redirections http. Il est uniquement utilisé pour désactiver les avertissements émanant des requêtes dans le cadre de ce script ;
- **Requests** : Module permettant d'utiliser le protocole http, d'accéder à l'API de l'Appstore et de faire des requêtes pour récupérer des données spécifiques ;
- **Json** : Le module JavaScript Object Notation (JSON) permet principalement d'encoder ou de décoder des objets. Dans ce script, il est utilisé pour stocker les données des applications sous le format JSON, charger les fichiers qui sont enregistrés sous ce même format communiquer d'éventuelles erreurs ;
- **Numpy** : Package permettant de manipuler des tableaux multi-dimensionnels et d'effectuer des fonction mathématiques ;
- **Time** : Module permettant de travailler avec toutes les fonctions liées au temps ; en l'occurrence il est utilisé pour espacer chaque requête de 2 secondes ;
- **Pandas** : Package permettant de travailler avec différents types de structures de données ; il est principalement utilisé dans ce script pour inscrire les données dans des dataframes et les manipuler ;
- **PCA** : Fonction de Sklearn permettant d'exécuter une réduire les dimensions d'un dataset de manière linéaire ;
- **TSNE** : Outil de Sklearn permettant de visualiser des données haute-dimension ;

— **KMeans** : Fonction de Sklearn permettant de réaliser des clusters ;

— **Pairwise_distances_argmin_min** : Fonction de Sklearn permettant de calculer la distance minimum entre un point et un ensemble de points ;

— **Matplotlib.pyplot** : Module de la librairie matplotlib utilisé pour faire des graphes en 2 dimensions ;

— **Figure** : Fonction de matplotlib.pyplot permettant de créer des figures et de les paramétrer ;

— **Train_test_split** : Fonction de Sklearn permettant de séparer les données en un ensemble d'entraînement et un autre ensemble de test ;

— **RandomForestClassifier** : Fonction de Sklearn permettant de construire des arbres de décision en vue de réaliser des prédictions de classifications ;

— **GridSearchCV** : Fonction de Sklearn permettant de procéder à une sélection de modèle optimal sur base d'un choix de différents paramètres ;

— **Confusion_matrix** : Fonction de Sklearn permettant de développer une matrice de confusion pour déterminer la performance d'un modèle de prédiction ;

— **F1_score** : Fonction de Sklearn retournant le score de performance d'un modèle de prédiction ;

Dans un premier temps, le type de requête implémenté pour collecter les données est le « lookup » qui permet de récupérer des informations techniques sur base de l'ID de l'artiste. En l'occurrence, l'ID de l'artiste correspond à l'identifiant de l'éditeur de l'application en question. Ce lookup est réalisé en démarrant d'une valeur d'ID choisie aléatoirement qui est 500000000 et en répétant la requête 100000000 fois en incrémentant la valeur de l'ID d'une unité à chaque fois. La requête lancée avec le module Requests a comme paramètres : la valeur de l'ID au moment de la boucle, le type de résultat attendu ; le type logiciel, et la valeur limite du nombre de résultats qui est fixée à 200 par requête. Ensuite, la fonction raise_for_status() est utilisée pour soulever une erreur http lorsqu'elle survient et, si celle-ci ne survient pas, le résultat de chaque requête est converti au format Json. Pour chaque requête, si le résultat correspond aux paramètres définis, alors les données du résultat sont rajoutées au tableau destiné à contenir les données relatives aux applications. Une fois cela fait, l'ID est incrémenté d'une unité et le système prend un repos de 2 secondes avant de lancer la requête suivante via la fonction sleep() pour ne pas être surchargé en poursuivant la boucle. Le processus continue de cette manière jusqu'à la fin de la boucle. Enfin, les données sont enregistrées dans un fichier au format JSON.

Durant ce recueil de données, différents types d'erreurs peuvent survenir, c'est pourquoi un mécanisme de gestion des erreurs de requêtes a été mis en place en vue de ne pas perturber le déroulement du script. Ce mécanisme est structuré en déclarant les gestions d'erreurs en allant d'un niveau d'abstraction faible à un niveau d'abstraction plus élévé pour que les erreurs les plus spécifiques ne soient pas masquées par les plus générales. Lorsque ces erreurs sont attrapées,

elles sont affichées à l'écran et ensuite, la boucle continue à tourner à partir de l'ID suivant. Les erreurs attrappées sont les erreurs qui arrivent lorsque :

- La réponse http à la requête est invalide : HTTPError ;
- Il y a un problème de connection : ConnectionError ;
- Une requête expire : Timeout ;
- Tout autre type d'exception soulevé par Requests se produit : RequestException ;

Dans un second temps, compte-tenu du temps que prend la récupération des données en naviguant d'IDs en IDs et le peu de données propices à être traitées recueillies, le processus a dû être arrêté avant la fin de la boucle et un autre mode opératoire a dû être adopté. En se basant sur le processus d'optimisation de magasins d'applications (ASO), une série de mots-clés a été recensée en vue d'implémenter des requêtes de type « search ». Les 500 meilleurs mots-clés utilisés par les éditeurs d'applications en 2019 pour gagner en visibilité ont étés trouvés sur [2] . Lorsqu'une requête de ce type est effectuée, les paramètres « term » et « country » sont obligatoires. Le premier paramètre a été rempli avec les 250 premiers mots-clés recensés et le deuxième a été fixé à US car seuls les valeurs des ratings US sont disponibles. En plus de ça, le paramètre « entity » est rempli avec software et enfin, ici encore, la limite du nombre de résultats attendu est établie à 200 résultats à la fois. En dehors de ces paramètres et du fait que les requêtes "search" bouclent jusqu'à avoir parcouru l'ensemble des mots-clefs, le mode de fonctionnement de cette méthode adopte les mêmes caractéristiques que le précédent. Le mécanisme de recensement des résultats, d'enregistrement de ceux-ci ainsi que les mécanismes de gestion des exceptions sont les mêmes.

4.2 Traitement des données

4.2.1 Prétraitement des données

Le format du dataset recueilli a été changé en dataFrame qui constitue un format plus flexible pour le traitement de données multi-dimensionnelles. Une fois le dataset construit, les données sont nettoyées en supprimant les duplicats, les données non-relatives à des applications et celles qui n'ont pas été notées car celles-ci n'apporteront aucune information pour répondre à la question de recherche.

Une fois les observations nettoyées, les variables, aussi appelées caractéristiques, doivent être traitées à leur tour. En effet, un ensemble de données ayant une quantité moindre de variables mène à des modèles qui apprennent à une vitesse plus élevée, c'est pourquoi il est important d'éliminer les variables qui n'apportent pas ou très peu d'information pour répondre à la question de recherche. Suite au recueil de données, 79 caractéristiques différentes ont été trouvées pour les applications du dataset ; celles-ci sont disponibles à l'Annexe A. La gestion de ces caractéristiques se fait par le biais de plusieurs opérations :

— Les caractéristiques dont les valeurs sont toutes nulles sont supprimées car elles n'apporteront pas d'information pour pouvoir répondre à la question de recherche. Dès lors, les caractéristiques 'amgArtistId', 'artistIds', 'artistLinkUrl', 'artistType', 'artworkUrl30', 'artworkUrl600', 'collectionArtistId', 'collectionArtistName', 'collectionArtistViewUrl', 'collectionCensoredName', 'collectionExplicitness', 'collectionHdPrice', 'collectionId', 'collectionName', 'collectionPrice', 'collectionType', 'collectionViewUrl', 'copyright', 'country', 'discCount', 'discNumber', 'feedUrl', 'isStreamable', 'longDescription', 'previewUrl', 'shortDescription', 'trackCount', 'trackExplicitness', 'trackHdPrice', 'trackHdRentalPrice', 'trackNumber', 'trackPrice', 'trackRentalPrice' et 'trackTimeMillis' sont retirées de l'ensemble de données qui n'est donc plus composé que de 45 caractéristiques.
— Les caractéristiques dont toutes les valeurs sont les mêmes sont supprimées car elles n'apporteront pas d'information pour pouvoir répondre à la question de recherche. Dès lors, les caractéristiques 'appletvScreenshotUrls', 'currency', 'isVppDeviceBasedLicensingEnabled', 'kind' et 'wrapperType' sont retirées de l'ensemble de données restant qui n'est donc plus composé que de 40 caractéristiques.
— Les caractéristiques dont toutes les caractéristiques sont différentes sont supprimées car elles n'apporteront pas d'information pour pouvoir répondre à la question de recherche. Dès lors, les caractéristiques 'bundleId', 'artworkUrl100', 'artworkUrl512', 'artworkUrl60', 'trackId' et 'trackViewUrl' sont retirées de l'ensemble de données restant qui n'est donc plus composé que de 34 caractéristiques.
— Certaines des caractéristiques restantes apportent les mêmes informations que d'autres étant donné qu'elles sont parfaitement, ou du moins très fortement correlées. Conserver ces variables en même temps dans l'ensemble de données final risquerait de rendre le modèle final moins performant car ça pourrait masquer des interactions existantes entre les autres caractéristiques et diminuer l'interprétabilité du modèle. Dès lors, certaines caractéristiques ont été retirées au profit d'autres ; les noms ont généralement été choisis par rapport aux ID et ce, par soucis d'interprétabilité. La variable 'genres' a été préférée à 'genreIds', 'artistName' a été préféré à 'artistId', 'primaryGenreName' a été préféré à 'primaryGenreId', 'userRatingCount' a été préféré à 'userRatingCountForCurrentVersion', 'averageUserRating' a été préféré à 'averageUserRatingForCurrentVersion', 'formattedPrice' a été préféré à 'price' et 'trackName' a été préféré à 'trackCensoredName'.
— Compte-tenu du fait que les variables restantes sont de différents types, elles doivent être changées en variables catégoriques pour pouvoir être encodées dans un format comparable par après. Certaines de ces variables sont facilement catégorisables mais d'autres nécessitent toutefois un traitement supplémentaire :
    — Les caractéristiques 'contentAdvisoryRating', 'features', 'formattedPrice', 'isGameCenterEnabled', 'trackName', 'trackContentRating' et 'primaryGenreName' ont simplement été transformées en variables

catégoriques pour utilisation ultérieure car chacune n'était composée que de peu de catégories différentes ;

- Les caractéristiques 'advisories', 'genres', 'languageCodesISO2A' et 'supportedDevices' sont des variables multicatégoriques. Par conséquent, il n'est pas possible de les encoder telles quelles :
  - La caractéristique 'advisories' a été traitée de 2 manières. Dans un premier temps, les différentes catégories disponibles dans l'ensemble des observations a été recensé et chacune d'entre-elles a été transformée en caractéristique à part entière et donc en 23 caractéristiques binaires. Ensuite, le nombre d'éléments 'advisories' pour chaque observation a été calculé et a été intégré comme caractéristique supplémentaire sous forme de classe ;
  - La caractéristique 'genres' a été traitée en recensant les différentes catégories disponibles dans chacune des observations et chacune d'entre elles a été transformée en caractéristique à part entière. Ceci donnant lieu au remplacement de la caractéristique 'genres' par 53 caractéristiques binaires ;
  - La caractéristique 'languageCodesISO2A' a également été traitée de 2 manières. Dans un premier temps, les différentes catégories disponibles dans l'ensemble des observations a été recensé et chacune d'entre-elles a été transformée en caractéristique à part entière et donc en 148 caractéristiques binaires. Ensuite, le nombre d'éléments 'languageCodesISO2A' pour chaque observation a été calculé et a été intégré comme caractéristique supplémentaire sous forme de classe ;
  - La caractéristique 'supportedDevices' a été traitée en recensant les différentes catégories disponibles dans chacune des observations et chacune d'entre elles a été transformée en caractéristique à part entière. Ceci donnant lieu au remplacement de la caractéristique 'supportedDevices' par 81 caractéristiques binaires ;
- Compte-tenu du fait que les variables 'artistViewUrl', 'ipadScreenshotUrls', 'screenshotUrls' et 'sellerUrl' sont toutes des caractéristiques uniquement composées d'urls, les conserver dans ce format pour les transformer en variables binaires pour chacune des valeurs par après n'apportera que très peu d'informations et rendra la matrice finale que plus "sparse". Dès lors, au lieu de simplement les supprimer, chacune de ces caractéristiques a été transformée en caractéristique binaire témoignant de la présence ou non de l'url pour chacune des observations ;
- Dans la même lignée, la variable 'releaseNotes' a également été transformée en caractéristique binaire indiquant la présence ou non de notes pour chaque observation. En effet, cette variable est composée de données qui sont différentes les unes des autres d'une observation à une autre. Cependant, ces notes ne sont pas publiées pour toutes

les observations, d'où l'intérêt de traiter cette caractéristique de cette manière ;

— Le type de la caractéristique 'releaseDate' a été changé en datetime. Sur base de ce changement, les mois de cette variables ont été récupérées pouvoir établir si la sortie à un mois particulier a une influence sur le rating de l'application ;

— La variable 'minimumOsVersion' a été traitée en récupérant la première valeur du numéro de version pour pouvoir créer des catégories moins éparpillées lorsque les caractéristiques seront séparées en caractéristiques binaire. Dès lors, pour chaque observation, la catégorie dont fait partie la valeur lui est assignée. De cette manière, des variables binaires sont crées sur base de si les valeurs font partie d'une de ces catégories ou si la version n'est pas mentionnée ;

— Les valeurs de la caractéristique 'fileSizeBytes' sont regroupées en classes selon si celles-ci sont petites, moyennes ou grandes. Une fois regroupées, ces classes sont elles aussi transformées en variables binaires ;

— La caractéristique 'index' a été supprimée car aucune documentation claire sur sa signification n'a été trouvée ;

— Les caractéristiques 'userRatingCount', 'version', 'currentVersionReleaseDate' ont été supprimées car elles ne permettent pas d'émettre de recommandation ;

— Dans l'ensemble de données final, seules les classes recensant le nombre de langues supportées par l'application ont été intégrées car conserver toutes les différentes langues diminuait les performances du modèle ;

— La caractéristique 'averageUserRating' est utilisée comme variable output pour le modèle. Cette caractéristique reprend la moyenne des ratings donnés par les utilisateurs. Pour que les interprétations se fassent plus facilement, les valeurs de cette caractéristique ont été regroupées en classes. Ces classes sont les suivantes :

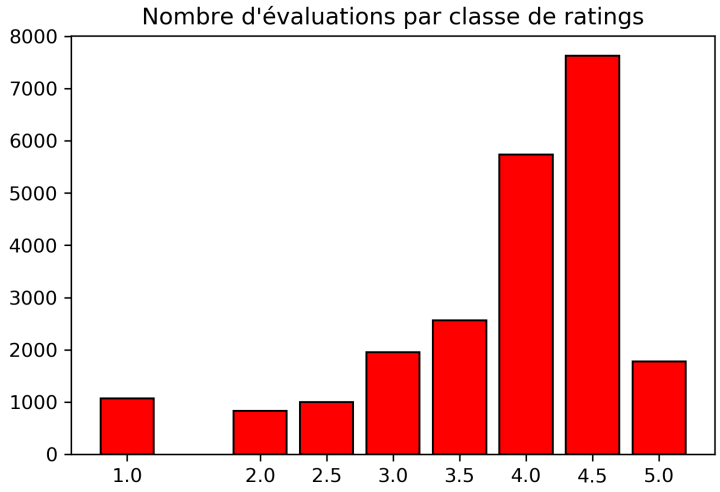

Figure 1 : Distribution du nombre d'évaluations par ratings

- 1.0 : Pour les applications ayant un rating compris entre 1 compris et 2 non-compris ;
- 2.0 : Pour les applications ayant un rating compris entre 2 compris et 2,5 non-compris ;
- 2.5 : Pour les applications ayant un rating compris entre 2,5 compris et 3 non-compris ;
- 3.0 : Pour les applications ayant un rating compris entre 3 compris et 3,5 non-compris ;
- 3.5 : Pour les applications ayant un rating compris entre 3,5 compris et 4 non-compris ;
- 4.0 : Pour les applications ayant un rating compris entre 4 compris et 4,5 non-compris ;
- 4.5 : Pour les applications ayant un rating compris entre 4,5 compris et 5 non-compris ;
- 5.0 : Pour les applications ayant un rating de 5 ;

La raison de la différentiation des milieux de classes à partir d'un rating est le fait qu'il existe une réelle différence entre une application ayant un rating moyen de 2 et qui n'est donc globalement pas bien reçue, un rating moyen de 2,5 et qui peut donc toujours soit s'empirer, soit s'améliorer et un rating de 3 qui est au-dessus de la moyenne et est considéré comme correcte. Dès lors, dans la même lignée, une application avec un rating de 4 et une autre avec un rating de 4,9 n'auront pas la même performance et ne devraient donc pas être classées dans la même catégorie. En termes de distribution des ratings recensés, il est également clair que la distribution des ratings entre les applications ayant un "mauvais" rating n'est pas la même que

celle de celles qui ont des ratings plus élevés (Figure 1).
Le cas des applications ayant été évaluées à 5 est un cas à part, c'est pourquoi il constitue une classe à part entière. [15] ;

4.2.2 Traitement global des données

Le traitement de la totalité de l'ensemble de données se fait par le biais d'une visualisation des données sous forme de clusters pour mener à la récupération des caractéristiques les plus importantes du modèle prédictif global.

Une extraction de caractéristiques par ACP et une réduction de dimensions par t-SNE (t-distributed Stochastic Neighbor Embedding) sont menées en vue de réaliser la visualisation des données. Le choix du nombre de composantes à extraire lors de l'ACP se fait sur base d'un graphe reportant la variance expliquée par composante principale. Cette réduction de dimensions permet d'exécuter plus aisément la seconde réduction de dimension qu'est le t-SNE. Cette méthode permet de réduire les dimensions de l'ensemble de données initial à un ensemble à 2 dimensions pour pouvoir faciliter leur visualisation. L'algorithme du t-SNE a pour but de représenter correctement des données à haute-dimension dans des dimensions moins élevées ; c'est donc un outil de visualisation qui procède en diminuant les dimensions de l'ensembles de données initial. Dès lors, avec cette méthode, les observations qui sont proches initialement sont supposées le rester lorsque les dimensions diminuent. Le paramètre du t-SNE qui est optimisé dans ce cas-ci est la perplexité qui est un paramètre lié au nombre de plus proches voisins utilisés pour visualiser les données. Dès lors, la structure de la visualisation des données diffère selon la valeur de ce paramètre. Celui-ci est testé pour les valeurs 5, 15, 30, 45 et 50.

Les données à deux dimensions générées à l'aide du t-SNE sont clusterisées grâce aux prédictions issuent de l'algorithme des K-means. Celui-ci permet de définir des clusters sur base d'un ensemble de données à prédire, qui est celui issu du t-SNE, et d'un nombre de clusters spécifique. Le but de la classification non-supervisée est de s'abstraire des connaissances a priori sur la classification attendue et de laisser les données se classer naturellement sur base de propriétés similaires. Dès lors, le choix du nombre de clusters à réaliser se fait en se basant sur le graphe de l'évolution de l'inertie par rapport au nombre de clusters, allant de 1 à 9. L'inertie définit à quel point les observations qui se trouvent au sein d'un même cluster sont éloignées les unes des autres. Le but étant de minimiser l'intertie intra-cluster, le k optimal est celui pour lequel l'ajout d'un cluster supplémentaire ne diminue plus significativement la variance de l'inertie. Enfin, une visualisation des données est réalisée sur base des données réduites et des clusters obtenus.

4.2.3 Traitement des données par méthode de classification

Compte-tenu du fait que le but de cette étude est de récupérer les caractéristiques menant au succès des applications mobiles, la méthode de classification choisie est l'utilisation de Forêts aléatoires pour pouvoir récupérer l'importance

des caractéristiques. Cette classification se fait en utilisant les données brutes et ne fait donc pas usage des données réduites par ACP. La raison est que la réduction de dimension par ACP donne lieu à une transformation des données initiales pour perdre un minimum d'information tout en réduisant la dimension de l'ensemble de données. Sachant que l'interprétaion des caractéristiques est l'objet de cette étude et que les données issues d'une transformation par ACP sont plus difficilement interprétables, les données brutes ont été préférées aux données réduites. Dès lors, la méthode des forêts aléatoire a été choisie car c'est une méthode qui n'utilise que les caractéristiques qui influencent la variance de l'output à prédire et qui est généralement plus robuste que les arbres de décision, bien qu'elle ait moins de facilité d'interprétation des caractéristiques par classe. Un 'GridSearchCV' est également utilisé pour retourner le modèle optimal pour faire les meilleures prédictions qui soient. Les paramètres parmis lesquels GridSearchCV peut choisir sont les suivants :

- Le nombre d'estimateurs, c'est-à-dire d'arbres dans la forêt. Ce nombre peut être de 5, 10, 15, 25, 35, 50, 75 ou 100 ;
- La fonction mesurant la qualité d'une séparation peut être évaluée soit sur base du critère "gini" mesurant l'impureté, soit sur base du critère "entropy" qui mesure le gain d'information ;
- Le paramètre "bootstrap" qui est binaire. Si sa valeur est True alors des échantillons bootstrap sont utilisés pour construire les arbres, sinon l'entièreté de l'ensemble de données est utilisé ;
- Le nombre de caractéristiques à considérer lors de la recherche du meilleur séparateur ; il peut être "sqrt" ou "auto" ;

4.3 Analyse des données

Ce mémoire vise à donner des recommandation managériales aux futurs développeurs d'applications mobiles, comme mentionné précédemment. Des recommandations générales seront présentées dans un premier temps sur base d'une évaluation de l'ensemble des données. Cependant, les attentes des utilisateurs qui décideront du succès ou non de l'application ne sont pas les mêmes d'une catégorie d'application à une autre. C'est pourquoi, dans un deuxième temps, les données seront séparées selon les genres principaux des applications fournis par la variable 'primaryGenre'. Il faut noter que le genre "Magazines & Newspapers" n'a pas pu être traité ici car l'ensemble de données d'entrainement n'était constitué que de 2 instances. Enfin, une analyse inter-catégories sera menée pour discuter des dimensions récurrentes et lier celles-ci aux recommandations générales émises.

La performance du modèle de classification établit est évaluée dans un premier temps sur base d'une matrice de confusion. Celle-ci permet de détecter à quel niveau les erreurs de classification se font. Les matrices propres à chaque genre d'application sont disponibles dans le script python disponible à l'annexe B. Ensuite, la métrique utilisée pour mesurer le taux de classification optimal

des observations est le f1 Score. Cette métrique est choisie car elle permet de calculer le score de classification tout en pondérant la précision qui est une mesure de l'exactitude ou de la qualité et le rappel qui est une mesure de l'exhaustivité ou de la quantité dans les résultats. Globalement, compte-tenu du fait que 8 catégories de ratings ont été définies, la probabilité de classer correctement une application de manière aléatoire est de 1 sur 8 et donc de 0,125. Il est donc question de trouver des modèles ayant des scores permettant de surpasser ce benchmark.

5 Résultats

5.1 Description des données

Durant le recueil de données, la méthode "Lookup" a permis de rassembler 768 observations en plusieurs jours d'exécution alors que la méthode "Search" via le processus ASO, elle, a permis de rassembler 37947 observations en seulement quelques minutes. Toutes ces données ont dès lors mené à un ensemble de données comprenant 38715 observations et 79 variables à traiter. Suite au premier nettoyage de données, l'ensemble de données est alors composé de 22545 observations. Le traitement des caractéristiques a donné lieu à un ensemble de données composé 242 caractéristiques. Celles-ci sont issues des caractéristiques initiales catégorisées et ont parfois fait l'objet de transformations :

- contentAdvisorRating : définit l'âge à partir duquel le contenu est approprié selon l'évaluateur de contenus ;
- features : comprend le fait que l'application soit activée pour le Game Center, optimisée pour tous les appareils iOS, détienne les deux caractéristiques ou aucune des deux ;
- formattedPrice : définit le prix formatté à la devise ;
- isGameCenterEnabled : définit si l'application est activée pour le Game Center ;
- définit l'âge à partir duquel le contenu est approprié ;
- genre : liste les différentes catégories dans lesquelles l'application fait partie ;
- supportedDevices : liste les appareils sur lesquels l'application peut être utilisée ;
- sellerUrl : définit si l'url de l'entreprise détentrice de l'application est mentionnée ;
- screenshotUrls : définit si les urls des screenshot des interfaces sont disponibles ;
- releaseNotes : définit la présence, ou non de notes soumises par le développeur pour la dernière version ;
- releaseDate : définit le mois de lancement de l'application ;
- minimumOsVersion : définit la version d'OS minimum pour pouvoir utiliser l'application ;
- ipadScreenshotUrls : définit si les urls des screenshot des interfaces sur iPad sont disponibles ;
- artistViewUrl : définit si l'url du développeur de l'application est mentionnée ;
- advisories : liste les différents types de publicités présentes dans l'application et définit leur quantité ;
- fileSizeBytes : définit la taille de l'application ;

— languageCodesISO2A : définit le nombres de langues dans lesquelles l'application est disponible ;

— primaryGenreName : définit le genre principal de l'application et sert de critère de séparation de l'ensemble de données en catégories à analyser. Cette caractéristique est donc utilisée pour l'analyse des données globales mais pas pour l'analyse par catégorie car la variance est de 0 dans ces cas ;

— averageUserRating : définit le rating moyen obtenu par l'application et sert d'output à prédire.

Les modèles qui ont été implémentés dans le cadre de ce travail ont pour but de permettre la prédiction des classifications d'applications futures sur base de certains facteurs techniques sur l'échelle de classification définie dans la section 3.2.1. L'analyse des prédictions qui suit se fera donc uniquement sur les critères menant à un rating allant de 4 à 5.

5.2 Analyse des données globales

5.2.1 Clustering des données

Le clustering des données a été réalisé suite à une double réduction de dimensions. La première a été faite par ACP en se basant sur le graphe de la variance expliquée par composante visible sur la Figure 2. Celui-ci montre qu'avec seulement 50 caractéristiques, plus de 88% de la variance est expliquée.

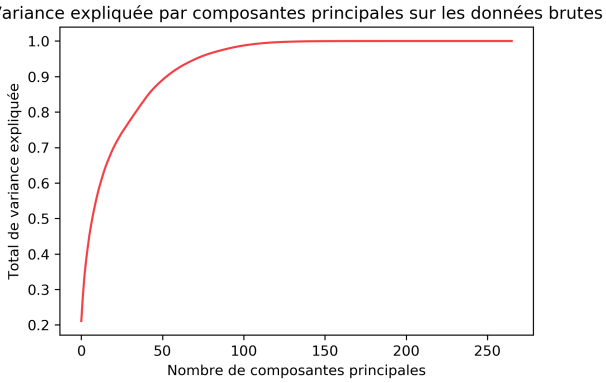

Figure 2 : Variance expliquée par composante

Dès-lors, ces 50 caractéristiques sont choisies pour définir le nombre de composantes à utiliser dans le premier ensemble de données réduit pour réaliser le clustering et la seconde réduction de dimensions. Sur base de cet ensemble, plusieurs autres ensembles de données sont construits pour le réduire à un ensemble à 2 dimensions à l'aide du t-SNE. Ces différents ensembles sont construits sur base des valeurs de perplexité définies au point 4.2.2. Ils sont chacuns testés avec différents modèles de clustering de K-Means. Le graphe de l'évolution de l'inertie par rapport au nombre de clusters est disponible sur la Figure 3. En se

basant sur la méthode du coude, le graphe indique que les valeurs de k optimales sont 3, 4 ou 5. Ces valeurs de k constituent les nombres de clusters posés dans les différents modèles de k-means testés avec les données réduites par t-SNE.

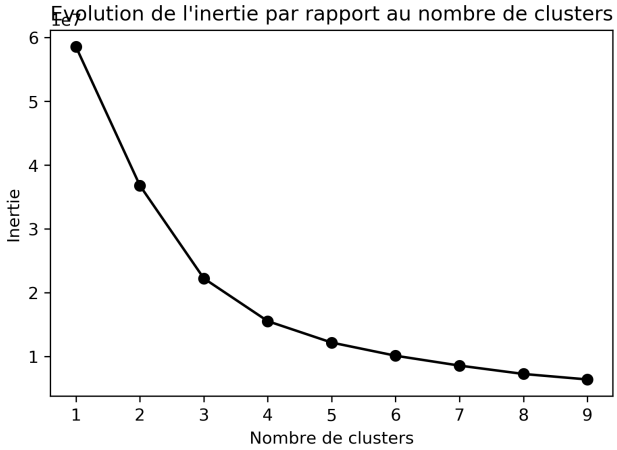

Figure 3 : Inertie par rapport au nombre de clusters

Le modèle sélectionné est celui qui fournit la visualisation des clusters la plus pertinente. Ce modèle est un modèle composé de 4 clusters et qui est visualisé avec une une perplexité de 45 (Figure 4). La raison de ce choix est que les données visualisées avec ces paramètres sont séparées de manière plus homogène que les autres.

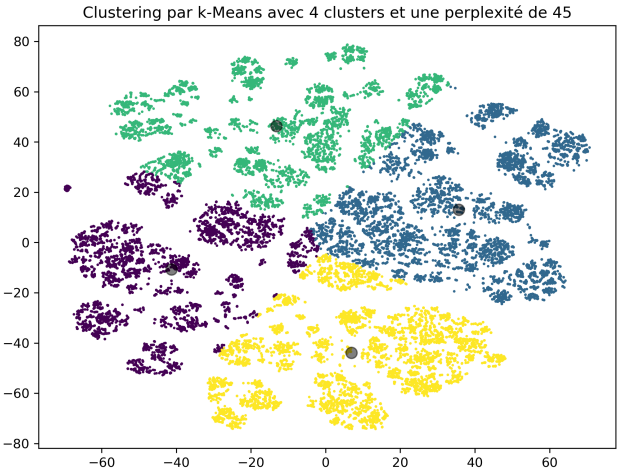

Figure 4 : Clustering des données par K-Means

La méthode des K-Means permet de regrouper les observations sur base de la similitude de leur propriétés qui est matérialisée par leur proximité dans un

espace de représentation. En déterminant les centres de clusters et en recherchant les instances les plus proches de ceux-ci, leurs ratings se sont révélés être 2.5, 3.5, 4 et 4. Ces valeurs indiquent donc que les données peuvent être séparées entre les applications à faible rating (2.5), les applications ayant un rating moyen (3.5) et les applications à rating élevé (4). Le fait que 2 catégories sur 4 reprennent des applications à rating élevé fait sens étant donné que les classes 4 et 4.5 constituent à elles-seules plus de la moitié des données de l'ensemble (Figure 1). Cet aspect est important car l'objet de cette étude est de trouver les déterminants des applications qui sont bien évaluées. Dès lors, entraîner des modèles de prédiction avec une quantité de données conséquente est important pour que les modèles construits aient une certaine validité.

Compte-tenu du fait qu'il a été établi que les données peuvent être séparées sur base de leur rating, une analyse plus en profondeur des déterminants de ces ratings peut être menée.

5.2.2 Critères généraux de succès

Il peut arriver qu'un entrepreneur souhaite lancer une application sans réellement savoir quelles catégories d'application sont les plus appréciées, quelles sont les dates optimales de lancement ou encore quelles données sont vraiment importantes à mentionner. C'est dans cette optique qu'une analyse des critères généraux de succès d'une application joue son role. Avoir une perspective globale de ce qui est attendu d'une application et de ce qui est valorisé dans celle-ci constitue un bon point de départ dans le processus de création d'une application.

Le modèle implémenté dans le cadre d'une application de n'importe quel genre a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : 'True', 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : '100'.

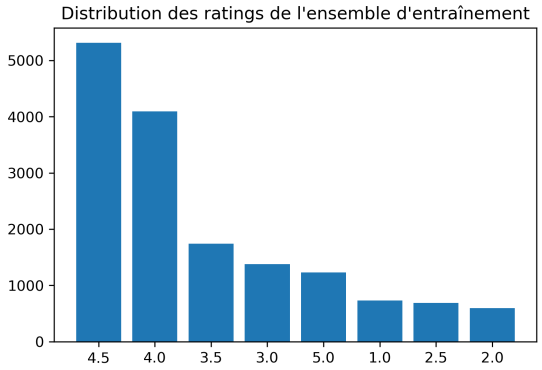

Figure 5 : Distribution de toutes les applications

De manière globale, le score de classification pour les données de test est de 0.32248 et le modèle utilise 232 caractéristiques. La matrice de confusion des

données test montre que le modèle permet essentiellement de prédire des applications se situant dans les classes 4 et 4.5, ce qui fait sens puisqu'il s'agit des classes majoritaires dans l'ensemble de données (Figure 5). La majorité des erreurs de classification se fait lorsque le modèle prédit la classification dans les classes 4 et 4.5 de données se trouvant réellement dans d'autres classes.

Features	Importance
sellerUrl_sellerUrl	0.028918
genres_'Entertainment'	0.028689
releaseDate_12	0.018798
releaseDate_4	0.018080
releaseDate_7	0.017960
releaseDate_3	0.017740
releaseDate_11	0.017497
releaseDate_9	0.017491
releaseDate_6	0.017087
supportedDevices_'Watch4-Watch4'	0.017068

Table 1 : Classement des caractéristiques générales les plus importantes

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la Table 1. Sur base de ces caractéristiques, l'un des aspects les plus importants est de publier l'url de l'entreprise fournissant l'application. En effet, de nos jours, la présence des logiciels malveillants est de plus en plus importante. C'est pourquoi il est très important de rassurer le consommateur en montrant que le fournisseur de l'application est sérieux et digne de confiance. Dès lors, publier l'url de l'entreprise permet aux consommateurs de se renseigner plus aisément et sans ambiguïté sur l'entreprise fournissant le service. De plus, les entreprises qui souhaitent lancer une application mais qui ne savent pas vers quelle catégorie se diriger devraient envisager de lancer une application de divertissement. En effet ces applications, avec l'accroissement de l'utilisation de smartphones, permettent aux individus d'utiliser leur téléphone pour regarder des films, séries ou autre depuis n'importe quel endroit. Ce type d'application est très important car, de nos jours, les individus sont constament en mouvement. Par conséquent, s'aligner aux programmes des chaînes télévisées devient de plus en plus difficile. Dès lors, fournir ce type de service gratuitement ou à petit prix est très attractif pour les consommateurs d'applications mobiles. Enfin, il est important de cibler les meilleurs moments pour sortir une application. En effet, la date de sortie de celle-ci a une grande part d'influence sur la manière dont elle est reçue par son public. Cibler les fêtes de fin d'années semble toujours être une bonne idée. Les appareils multimédias représentent une part importante des types de cadeaux offerts durant cette période. Par conséquent, il est intelligent de miser sur le fait que la réception de ces cadeaux engendre le téléchargement d'applications mobiles en tout genre.

5.3 Critères par catégorie d'application

La répartition des observations par catégories peut être observée sur la Figure 6. La catégorie majoritairement présente dans l'ensemble de données est "Games" ce qui fait sens puisqu'il s'agit de la catégorie majoritaire dans l'App

Store [12]. Dès lors, l'analyse dont elle fera l'objet dans la section suivante sera la plus représentative. En revanche, les analyses des catégories pourvues de moins d'observations telles que "Stickers", "Medical" ou encore "Graphics & Design" seront moins représentatives.

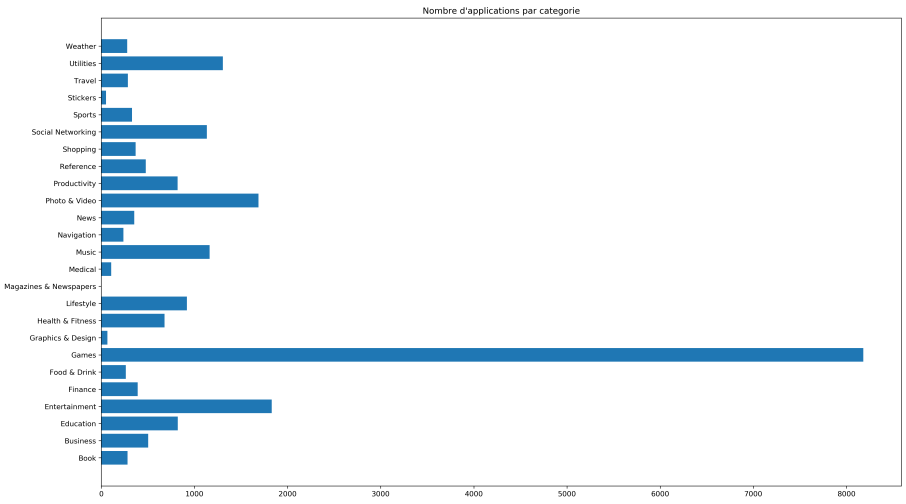

Figure 6 : Répartition du nombre d'applications par catégories

5.3.1 Navigation

Le modèle implémenté dans le cadre des applications dont le genre principal est "Navigation" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : False, 'criterion' : 'entropy', 'max_features' : 'sqrt' et 'n_estimators' : 50. Cependant, compte-tenu du score de classification retourné avec ces paramètres (0.11197), ces derniers ont été quelque peu ajustés pour pouvoir arriver à un score qui dépasse le benchmark. S'il y a plus de chances de faire de bons résultats avec une classification aléatoire qu'avec le modèle, l'implémentation du modèle s'en trouve, en effet, dénuée d'intérêt.

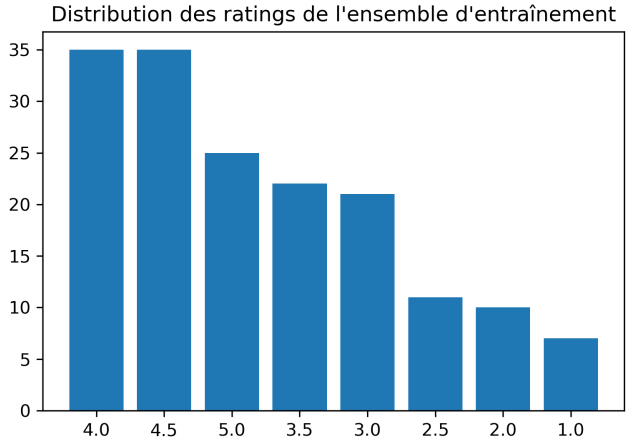

Figure 7 : Distribution des applications Navigation

Dès lors, le score de classification pour les données de test est de 0.1551 lorsque la valeur du paramètre 'bootstrap' est True et le nombre de caractéristiques utilisées est de 133. La matrice de confusion des données test montre que l'essentiel des classes prédites, peu importe la classe réelle, sont les classes 3.5, 4, 4.5 et 5, ce qui fait sens étant donné que ce sont les classes dans lesquelles il y a le plus de données d'entrainement pour cette catégorie (Figure 7).

Features	Importance
sellerUrl_sellerUrl	0.055594
genres_'Travel'	0.040087
genres_'Utilities'	0.037954
fileSizeClassMB_Big	0.036939
fileSizeClassMB_Small	0.033226
features_['iosUniversal']	0.031226
supportedDevices_'Watch4-Watch4'	0.029538
NbrlanguageCodesISO2A<5	0.028088
features_nan	0.027878
releaseDate_8	0.027547

Table 2 : Classement des caractéristiques les plus importantes de la catégorie Navigation

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 2. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types voyage ou utilitaire car ce sont toutes des catégories qui s'imbriquent bien et qui réalisent des fonctionnalités complémentaires. Ils devraient également faire en sorte qu'elle soit optimisée pour tous les appareils iOS et utilisable sur iWatch pour pouvoir l'utiliser à tout moment sans avoir à être en possession de ses autres appareils.

5.3.2 Jeux

Le modèle implémenté dans le cadre des applications dont le genre principal est "Games" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'entropy', 'max_features' : 'sqrt' et 'n_estimators' : 100.

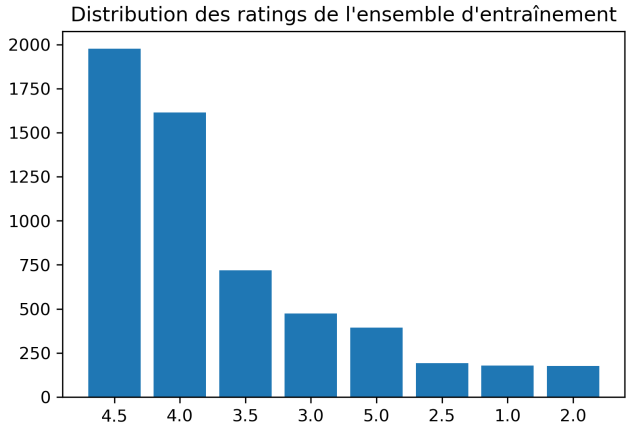

Figure 8 : Distribution des applications Jeux

De manière globale, le score de classification pour les données de test est de 0.34478 et le nombre de caractéristiques utilisé est de 202. La matrice de confusion des données test montre que l'essentiel des classes prédites, peu importe la classe réelle, sont les classes 3.5, 4 et 4.5, ce qui fait sens étant donné que ce sont les classes dans lesquelles il y a le plus de données d'entrainement pour cette catégorie (Figure 8). La matrice montre également que le modèle a beaucoup de mal à prédire la classe 5 ce qui peut être expliqué par le fait que c'est une classe qui a été entrainée avec peu de données, comparativement aux classes 4 et 4.5 par exemple.

Features	Importance
genres_'Entertainment'	0.040057
sellerUrl_sellerUrl	0.032693
genres_'Casual'	0.030363
releaseNotes_releaseNotes	0.027995
genres_'Action'	0.026503
genres_'Simulation'	0.024747
genres_'Adventure'	0.022264
genres_'Puzzle'	0.021162
genres_'Family'	0.020743
releaseDate_12	0.019345

Table 3 : Classement des caractéristiques les plus importantes de la catégorie Jeux

Les 10 caractéristiques qui influencent principalement la variance du rating pour

cette catégorie d'application sont reprises dans la table 3. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types divertissement, action, simulation, aventure, puzzle ou encore famille, car ce sont les applications qui fonctionnent le mieux, en termes de ratings. De plus, il est plus judicieux de lancer ce type d'application en décembre car c'est pendant les fêtes où les consommateurs reçoivent des nouveaux appareils et donc téléchargent de nouvelles applications. L'application devrait être accompagnée de notes dès la sortie d'une nouvelle version. En effet, les applications de jeux, plus souvent commentées par une communauté très impliquée, font régulièrement l'objet de mise à jour. C'est sûrement pour cette raison que l'on constate une augmentation des évaluations lors de nouvelles sorties, qui renseignent des modifications effectuées et montrent aux utilisateurs que leurs feed-back portent leurs fruits. Compte-tenu de la taille de l'ensemble de données pour cette catégorie, ces résultats sont assez représentatifs comparés aux autres catégories.

5.3.3 Référence

Le modèle implémenté dans le cadre des applications dont le genre principal est "Reference" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 75.

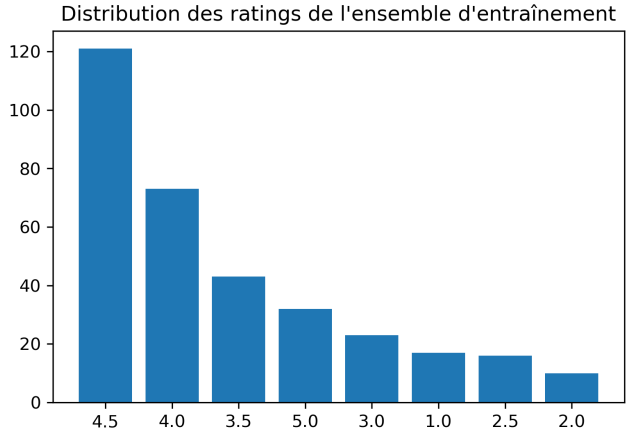

Figure 9 : Distribution des applications Référence

De manière globale, le score de classification pour les données de test est de 0.3212 et le nombre de caractéristiques utilisé est de 145. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classe 3.5, 4 et 4.5 au lieu des classes réelles, ce qui fait sens étant donné que ce sont les classes avec le plus de données dans l'ensemble de données d'entraînement (Figure 9). La matrice montre également que le modèle a du mal à trouver les déterminants de la classe 5 étant donné

qu'il se trompe régulièrement lorsqu'il doit prédire la classe et la prédit aussi alors qu'il ne devrait pas.

Features	Importance
sellerUrl_sellerUrl	0.034659
releaseNotes_releaseNotes	0.029168
fileSizeClassMB_Big	0.028890
fileSizeClassMB_Small	0.027653
releaseDate_4	0.024953
genres_'Utilities'	0.023527
minimumOsVersion_8	0.023208
supportedDevices_'Watch4-Watch4'	0.020434
NbrlanguageCodesISO2A<5	0.020069
fileSizeClassMB_Medium	0.019508

Table 4 : Classement des caractéristiques les plus importantes de la catégorie Référence

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 4. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type utilitaire car ce sont des catégories qui sont étroitement liées et qui permettent de rassembler, sur la même application, des éléments qui pourraient satisfaire les attentes de leur public cible. De plus, implémenter une application de ce type qui est utilisable sur iWatch la rendrait plus attractive pour les personnes qui souhaitent ne pas s'encombrer inutilement d'appareils. Enfin, créer une application qui ne requiert pas une version minimum d'OS trop poussée et qui tient ses utilisateurs au courant des mises à jour la rendrait également plus attractive.

5.3.4 Photos & Vidéos

Le modèle implémenté dans le cadre des applications dont le genre principal est "Photo & Video" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'entropy', 'max_features' : 'sqrt' et 'n_estimators' : 75.

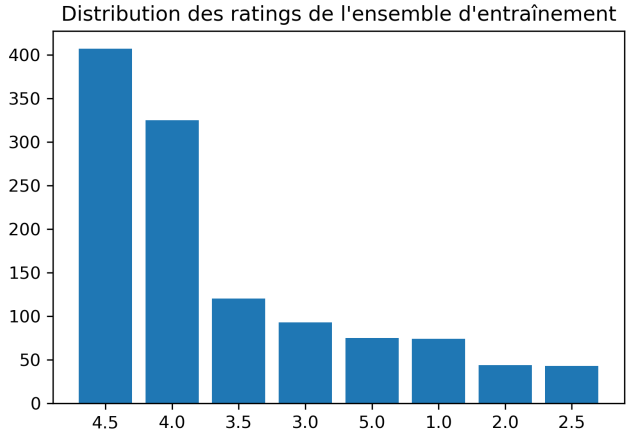

Figure 10 : Distribution des applications Photos et Vidéos

De manière globale, le score de classification pour les données de test est de 0.30398 et le nombre de caractéristiques utilisé est de 175. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classes 4 et 4.5 au lieu des classes réelles, ce qui fait sens étant donné que ce sont les classes avec le plus de données dans l'ensemble de données d'entraînement (Figure 10). La matrice montre également que le modèle a du mal à trouver les déterminants de la classe 5 étant donné qu'il se trompe régulièrement lorsqu'il doit prédire la classe et la prédit aussi alors qu'il ne devrait pas.

Features	Importance
sellerUrl_sellerUrl	0.048907
genres_'Entertainment'	0.041521
genres_'Utilities'	0.033684
fileSizeClassMB_Small	0.032822
fileSizeClassMB_Big	0.031236
genres_'Lifestyle'	0.030097
NbrlanguageCodesISO2A<5	0.028394
releaseNotes_releaseNotes	0.025876
releaseDate_5	0.023536
NbrlanguageCodesISO2A[5-20[	0.023232

Table 5 : Classement des caractéristiques les plus importantes de la catégorie Photos & Vidéos

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 5. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types divertissement, utilitaire ou encore lifestyle. En effet, ce sont des catégories qui sont toutes étroitement liées et qui pourraient faire usage de cette combinaison de fonctionnalités pour satisfaire les attentes de beaucoup de consommateurs de ce type d'application. De plus, implémenter une application qui est utilisable

dans un nombre important de langues serait une bonne idée compte-tenu du fait que ce n'est pas une application qui s'adresse à un public spécifique, géographiquement parlant. En outre, il semblerait que le mois optimal pour sortir ce type d'application soit le mois de mai, juste avant la période des vacances d'été qui est souvent synonyme de voyages. Enfin, il serait intéressant que ce soit une application moins lourde que celles faisant partie de la même catégorie, de manière à ce qu'elle s'en différencie.

5.3.5 Livres

Le modèle implémenté dans le cadre des applications dont le genre principal est "Book" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'entropy', 'max_features' : 'sqrt' et 'n_estimators' : 5.

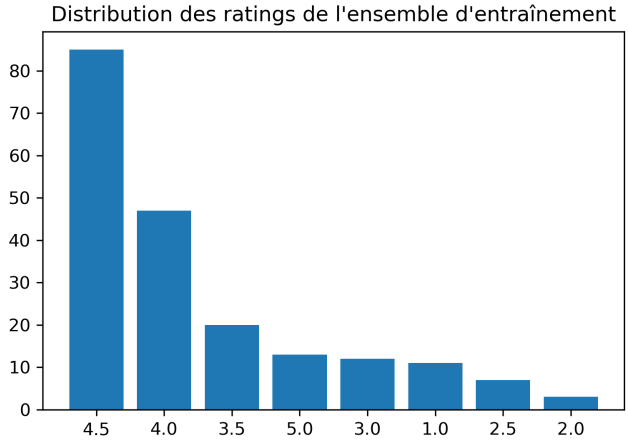

Figure 11 : Distribution des applications Livre

De manière globale, le score de classification pour les données de test est de 0.29256 et le nombre de caractéristiques utilisé est de 89. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classes 4 et 4.5 au lieu des classes réelles, ce qui fait sens étant donné que ce sont les classes avec le plus de données dans l'ensemble de données d'entraînement (Figure 11).

Features	Importance
sellerUrl_sellerUrl	0.032073
genres_'Entertainment'	0.031983
fileSizeClassMB_Small	0.031738
ipadScreenshotUrls_ipadScreenshotUrls	0.030459
NbrlanguageCodesISO2A[5-20[	0.030376
releaseDate_3	0.030089
genres_'Education'	0.028360
supportedDevices_'Watch4-Watch4'	0.028120
releaseDate_7	0.026214
genres_'Lifestyle'	0.024788

Table 6 : Classement des caractéristiques les plus importantes de la catégorie Livres

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 6. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type éducation ou lifestyle. En effet, ce sont des catégories qui permettent de générer une plus-value pour l'application que ce soit en termes éducatifs mais également dans la personnalisation que ça pourrait offrir. De plus, implémenter une application utilisable dans un nombre important de langues serait une bonne idée compte-tenu du fait qu'elle pourrait dès lors toucher un public plus large. De plus, il semblerait que les mois optimaux pour sortir ce type d'application soient les mois de mars, c'est-à-dire juste avant le début du printemps, et de juillet, c'est-à-dire en été. Ces deux moments sont ceux où les gens ont tendance à se rendre à la plage ou dans des parcs pour pouvoir se détendre et lire tranquillement. Enfin, puisqu'il s'agit d'une application utilisée sur le temps-libre, il serait intéressant qu'elle soit compatible avec des appareils tels que l'iPad ou l'iWatch.

5.3.6 Musique

Le modèle implémenté dans le cadre des applications dont le genre principal est "Music" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 50.

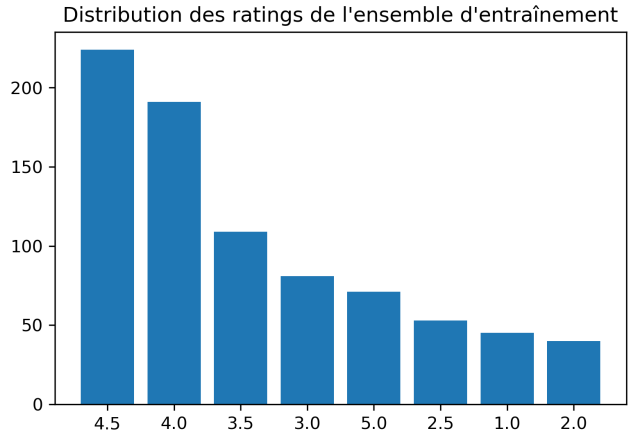

Figure 12 : Distribution des applications Musique

De manière globale, le score de classification pour les données de test est de 0.28811 et le nombre de caractéristiques est de 146. Encore une fois, la matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classes 4 et 4.5 au lieu des classes réelles,

ce qui fait sens étant donné que ce sont les classes avec le plus de données dans l'ensemble de données d'entraînement (Figure 12).

Features	Importance
sellerUrl_sellerUrl	0.044392
genres_'Entertainment'	0.039001
genres_'Utilities'	0.028938
NbrlanguageCodesISO2A<5	0.028900
fileSizeClassMB_Big	0.026387
genres_'Education'	0.025082
releaseDate_10	0.025034
fileSizeClassMB_Small	0.024531
supportedDevices_'Watch4-Watch4'	0.023601
releaseDate_11	0.023112

Table 7 : Classement des caractéristiques les plus importantes de la catégorie Musique

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 7. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types divertissement, utilitaires ou encore education et être accessible sur iWatch. En effet, ce sont des catégories qui permettraient de différencier l'application des autres plateformes de musique, en plus de la rendre accessible sans necessiter que les utilisateurs soient en possession de leurs autres appareils. Une autre manière de se différencier serait d'implémenter une application de musique qui ne soit pas trop lourde pour ne pas réduire excessivement l'espace de stockage de l'utilisateur. Enfin, lancer l'application en octobre ou en novembre, avant que le froid de l'hiver ne s'installe, lui permettrait d'entrer sur le marché à un moment efficace.

5.3.7 Productivité

Le modèle implémenté dans le cadre des applications dont le genre principal est "Productivity" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 75.

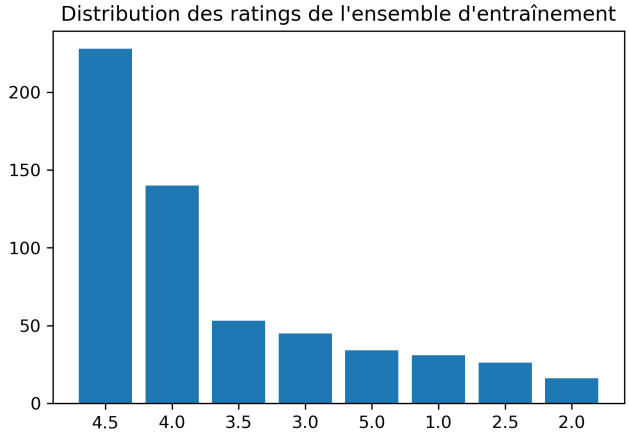

Figure 13 : Distribution des applications Productivité

De manière globale, le score de classification pour les données de test est de 0.25197 et le nombre de caractéristiques utilisé est de 154. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classes 4 et 4.5 au lieu des classes réelles, ce qui fait sens étant donné que ce sont les classes avec le plus de données dans l'ensemble de données d'entraînement (Figure 13).

Features	Importance
sellerUrl_sellerUrl	0.049014
genres_'Utilities'	0.043584
genres_'Business'	0.031027
supportedDevices_'Watch4-Watch4'	0.030493
NbrlanguageCodesISO2A<5	0.029128
NbrlanguageCodesISO2A[5-20[	0.027889
releaseDate_4	0.027245
fileSizeClassMB_Big	0.025483
fileSizeClassMB_Small	0.025075
fileSizeClassMB_Medium	0.022866

Table 8 : Classement des caractéristiques les plus importantes de la catégorie Productivité

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 8. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types utilitaires ou business. En effet, ce sont des catégories qui sont en lien étroit avec la productivité et qui, combinées, permettraient de rassembler, sur la même application, des éléments qui pourraient satisfaire les attentes de beaucoup de consommateurs friands de ce type d'application. De plus, implémenter une application qui est utilisable dans un nombre important de langues serait une bonne idée puisqu'elle ne s'adresse pas à un public spécifique, géographiquement parlant. Enfin, il serait intéressant qu'elle soit disponible sur iWatch pour garantir à l'utilisa-

teur une plus grande mobilité et une plus grande facilité dans la consultation de l'application.

5.3.8 Shopping

Le modèle implémenté dans le cadre des applications dont le genre principal est "Shopping" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 75.

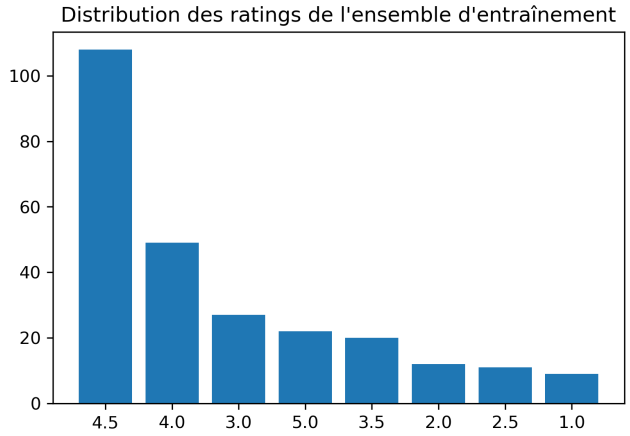

Figure 14 : Distribution des applications Shopping

De manière globale, le score de classification pour les données de test est de 0.35011 et le nombre de caractéristiques utilisé est de 122. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant la classe 4.5 au lieu des classes réelles, ce qui fait sens étant donné que c'est la classe avec le plus de données dans l'ensemble de données d'entraînement (Figure 14).

Features	Importance
genres_'Lifestyle'	0.047642
sellerUrl_sellerUrl	0.036753
fileSizeClassMB_Big	0.031846
fileSizeClassMB_Small	0.030195
releaseDate_9	0.029146
NbrlanguageCodesISO2A[5-20[	0.024765
releaseDate_2	0.023570
fileSizeClassMB_Medium	0.023513
minimumOsVersion_8	0.023336
NbrlanguageCodesISO2A<5	0.022704

Table 9 : Classement des caractéristiques les plus importantes de la catégorie Shopping

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 9. Sur base de ces carac-

téristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type lifestyle car ce sont 2 catégories qui sont étroitement liées ; cela permettrait de rassembler, sur la même application, des conseils mode, beauté et autre, tout en dirigeant l'utilisateur vers les produits à acheter. De plus, implémenter une application utilisable dans un nombre important de langues serait une bonne idée pour fournir ses services dans un espace géographique plus étendu. Enfin, il serait assez intéressant que l'application ne requiert pas une version d'OS minimum trop poussée pour qu'elle soit plus accessible, et par conséquent plus attractive, pour les utilisateurs de l'App Store.

5.3.9 Utilitaires

Le modèle implémenté dans le cadre des applications dont le genre principal est "Utilities" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'entropy', 'max_features' : 'sqrt' et 'n_estimators' : 100.

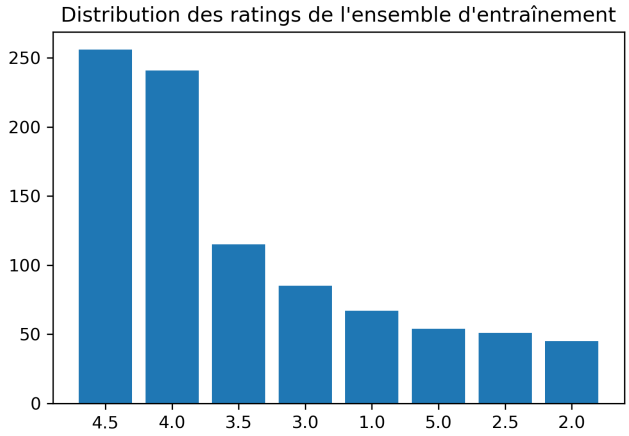

Figure 15 : Distribution des applications Utilitaires

De manière globale, le score de classification pour les données de test est de 0.26333 et le nombre de caractéristiques utilisé est de 189. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classes 4 et 4.5 au lieu des classes réelles, ce qui fait sens étant donné que ce sont les classes avec le plus de données dans l'ensemble de données d'entraînement (Figure 13).

Features	Importance
sellerUrl_sellerUrl	0.045304
genres_'Productivity'	0.036101
fileSizeClassMB_Small	0.030096
fileSizeClassMB_Big	0.028998
NbrlanguageCodesISO2A<5	0.028951
fileSizeClassMB_Medium	0.024202
genres_'Entertainment'	0.023993
supportedDevices_'Watch4-Watch4'	0.022757
features_nan	0.022399
minimumOsVersion_9	0.021775

Table 10 : Classement des caractéristiques les plus importantes de la catégorie Utilitaires

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 10. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type productivité. En effet, c'est une catégorie qui est étroitement liée aux utilitaires et leur combinaison permettrait de rassembler, sur la même application, des éléments qui pourraient satisfaire les attentes de beaucoup de consommateurs de ce type d'application. Dès lors, implémenter une application de ce type, utilisable à tout moment et à tout endroit permettant ainsi à l'utilisateur de ne pas se munir de ses appareils pour les utiliser, constituerait une plus-value. Enfin, il serait assez intéressant que l'application ne requiert pas une version d'OS minimum trop poussée pour qu'elle soit plus accessible, et par conséquent plus attractive, pour les utilisateurs de l'App Store.

5.3.10 Éducation

Le modèle implémenté dans le cadre des applications dont le genre principal est "Education" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 35.

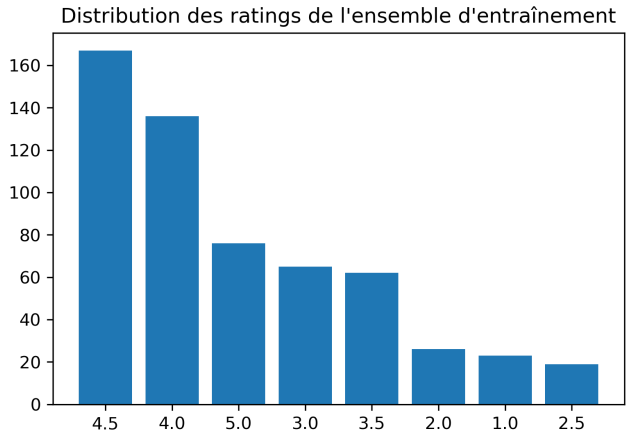

Figure 16 : Distribution des applications Éducation

De manière globale, le score de classification pour les données de test est de 0.23435 et le nombre de caractéristiques utilisé est de 159. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classes 4, et 4.5 au lieu des classes réelles, ce qui fait sens étant donné que ce sont les classes avec le plus de données dans l'ensemble de données d'entraînement (Figure 16). La matrice montre également que le modèle a du mal à prédire la classe 5 et a également tendance à prédire la classe 5 par erreur à la place des autres classe. Compte-tenu du fait que c'est la 3eme classe la plus peuplée des données d'entrainement, ces erreurs de classification ne font pas réellement sens.

Features	Importance
sellerUrl_sellerUrl	0.040014
fileSizeClassMB_Big	0.028666
genres_'Entertainment'	0.026763
genres_'Games'	0.026461
releaseDate_8	0.026136
releaseDate_11	0.025217
fileSizeClassMB_Small	0.024490
supportedDevices_'Watch5-Watch5'	0.024251
genres_'Family'	0.023760
releaseDate_7	0.023483

Table 11 : Classement des caractéristiques les plus importantes de la catégorie Éducation

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 11. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types divertissement, jeux ou encore famille. En effet, ce sont des catégories qui entretiennent un lien étroit avec l'éducation et qui permettent d'apprendre en s'amusant. De plus, lorsqu'il est possible d'utiliser ce type d'application depuis son iWatch, cela permet d'apprendre depuis n'importe quel endroit. Enfin, il semblerait que les mois optimaux pour sortir ce type d'application soient les mois de juillet et août, pour apprendre pendant les vacances, ou celui de novembre, juste avant les examens.

5.3.11 Santé & Fitness

Le modèle implémenté dans le cadre des applications dont le genre principal est "Health & Fitness" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 75.

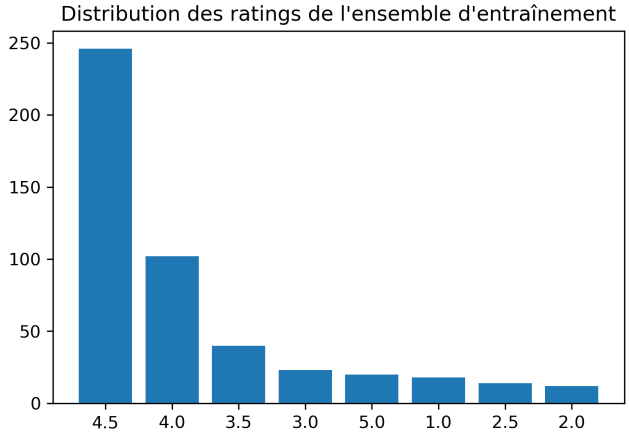

Figure 17 : Distribution des applications Santé & Fitness

De manière globale, le score de classification pour les données de test est de 0.38007 et le nombre de caractéristiques utilisé est de 175. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant la classe 4.5 au lieu des classes réelles, ce qui fait sens étant donné que c'est la classe avec le plus de données dans l'ensemble de données d'entraînement (Figure 17). La matrice montre également que le modèle ne parvient pas à prédire la classe 5 ce qui peut être dû à la quantité limitée de données dans cette classe durant l'entraînement du modèle.

Features	Importance
sellerUrl_sellerUrl	0.037873
genres_'Lifestyle'	0.037736
supportedDevices_'Watch4-Watch4'	0.029389
fileSizeClassMB_Small	0.028005
fileSizeClassMB_Big	0.027930
NbrlanguageCodesISO2A<5	0.022506
genres_'Medical'	0.022112
releaseDate_1	0.022060
releaseDate_6	0.021757
releaseDate_7	0.021415

Table 12 : Classement des caractéristiques les plus importantes de la catégorie Santé & Fitness

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 12. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types médical ou lifestyle. En effet, ce sont des catégories qui sont en lien étroit avec la santé et le fitness. Cela permettrait donc de créer une application plus complète, combinant les fonctionnalités initiales à des conseils médicaux ou d'adoption d'un style de vie plus sain de manière générale. Dès lors, implémenter une application accessible sur iWatch permettrait d'accéder à toutes ces fonctionnalités à tout moment, et, si l'application le prévoit, d'être notifié lorsque l'objectif journalier

physique n'a pas encore été atteint, ou lorsque les battements du coeurs sont anormalement rapides et qu'il faut consulter un médecin. Enfin, il semblerait que les mois optimaux pour sortir ce type d'application soient celui de janvier, pour se remettre en forme après les fêtes et ceux de juin et juillet, pour commencer l'été en bonne santé.

5.3.12 Sport

Le modèle implémenté dans le cadre des applications dont le genre principal est "Sports" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 75.

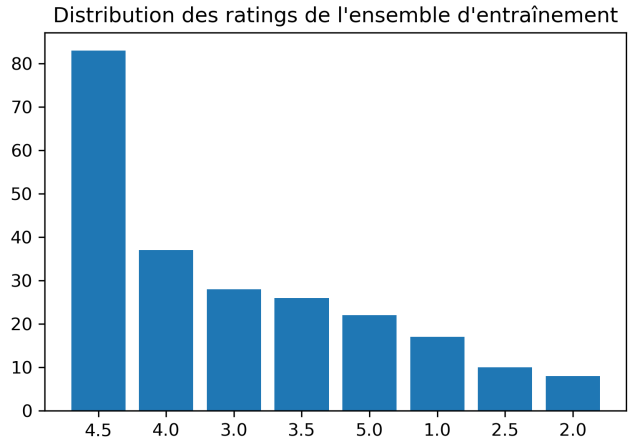

Figure 18 : Distribution des applications Sport

De manière globale, le score de classification pour les données de test est de 0.30715 et le nombre de caractéristiques utilisé est de 120. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant la classe 4.5 au lieu des classes réelles, ce qui fait sens étant donné que c'est la classe avec le plus de données dans l'ensemble de données d'entraînement (Figure 18). La matrice montre également que le modèle ne parvient pas à prédire la majorité des données de la classe 5 ce qui peut être dû à la quantité moins impoprtante de données dans cette classe durant l'entraînement du modèle.

Features	Importance
sellerUrl_sellerUrl	0.037221
genres_'Entertainment'	0.033340
fileSizeClassMB_Big	0.028851
supportedDevices_'Watch4-Watch4'	0.027956
releaseDate_9	0.027257
fileSizeClassMB_Medium	0.025932
fileSizeClassMB_Small	0.024805
releaseDate_8	0.024148
releaseDate_3	0.022060
releaseDate_2	0.020880

Table 13 : Classement des caractéristiques les plus importantes de la catégorie Sport

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 13. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type divertissement car avoir des fonctionnalités permettant de se divertir pendant une activité sportive complèterait l'application. Dans la même lignée, fournir une application de sport accessible sur iWatch, et donc permettant de ne pas devoir se déplacer avec ses autres appareils pour utiliser son application de sport, la rendrait encore plus optimale. Enfin, il semblerait que les périodes optimales pour lancer ce type d'application soient la fin de l'hiver, entre février et mars, lorsque le temps devient plus propice à l'exercice physique en extérieur, et la fin de l'été, entre août et septembre, lorsque les vacances se terminent et qu'il faut recommencer à adopter une routine de vie.

5.3.13 News

Le modèle implémenté dans le cadre des applications dont le genre principal est "News" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : False, 'criterion' : 'entropy', 'max_features' : 'sqrt' et 'n_estimators' : 25.

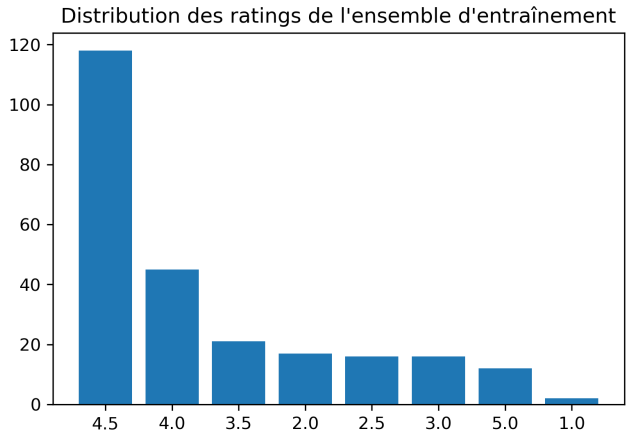

Figure 19 : Distribution des applications News

De manière globale, le score de classification pour les données de test est de 0.34005 et le nombre de caractéristiques utilisé est de 110. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant la classe 4.5 au lieu des classes réelles, ce qui fait sens étant donné que la classe 4.5 est celle avec le plus de données dans l'ensemble de données d'entraînement (Figure 19). La matrice montre également que le modèle ne parvient pas à prédire les données de la classe 5 ni la majorité des données de la classe 4, ce qui peut être dû à la quantité limitée de données dans ces classes durant l'entraînement du modèle.

Features	Importance
releaseDate_7	0.041464
sellerUrl_sellerUrl	0.036893
fileSizeClassMB_Big	0.035208
releaseDate_12	0.032462
advisories_'Infrequent/Mild Sexual Content and...	0.031715
fileSizeClassMB_Small	0.030225
releaseDate_9	0.029506
genres_'Weather'	0.028965
fileSizeClassMB_Medium	0.025932
advisories_'Infrequent/Mild Realistic Violence'	0.024928

Table 14 : Classement des caractéristiques les plus importantes de la catégorie News

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 14. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type météo pour compléter les informations avec la météo en lien avec les lieux où se déroule l'objet de l'information (canicules, inondations, etc). Enfin, il semblerait que les périodes optimales pour sortir ce type d'application soient au début de l'été (juillet), au début de l'année scolaire (septembre) et à la fin de l'année civile (décembre).

5.3.14 Business

Le modèle implémenté dans le cadre des applications dont le genre principal est "Business" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : False, 'criterion' : 'entropy', 'max_features' : 'sqrt' et 'n_estimators' : 35.

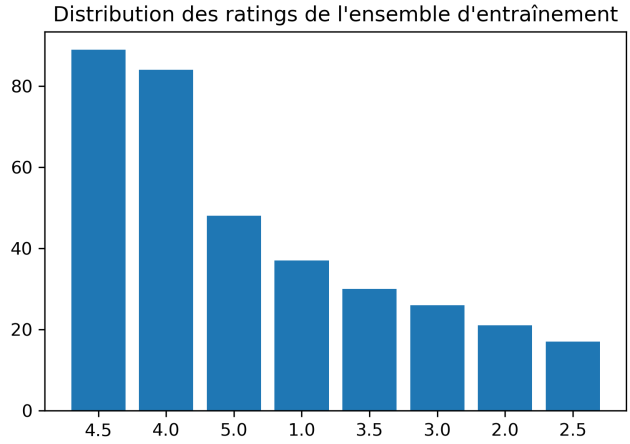

Figure 20 : Distribution des applications Business

De manière globale, le score de classification pour les données de test est de 0.23987 et le nombre de caractéristiques utilisé est de 159. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classes 4 et 4.5 au lieu des classes réelles, ce qui fait sens étant donné que ces classes sont celles avec le plus de données dans l'ensemble de données d'entraînement (Figure 20). La matrice montre également que le modèle ne parvient pas à prédire la majorité des données de la classe 5 ni la majorité des données de la classe 4, bien que ce soient les 2 classes avec le plus de données d'entraînement après la classe 4.5.

Features	Importance
sellerUrl_sellerUrl	0.048352
genres_'Productivity'	0.040328
NbrlanguageCodesISO2A<5	0.034120
fileSizeClassMB_Big	0.030251
releaseDate_2	0.030090
supportedDevices_'Watch4-Watch4'	0.028140
fileSizeClassMB_Small	0.027937
NbrlanguageCodesISO2A[5-20[	0.026698
supportedDevices_'Watch5-Watch5'	0.025671
releaseDate_7	0.024910

Table 15 : Classement des caractéristiques les plus importantes de la catégorie Business

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 15. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type productivité car il s'agit d'une catégorie en lien étroit avec le business et cela permettrait de rassembler, sur la même application, des éléments qui pourraient satisfaire les attentes de beaucoup de consommateurs de ce type d'application et donc de la rendre plus complète. En effet, fournir une application permettant de travailler

et qui permettrait également de maintenir la concentration de l'utilisateur en même temps serait très attractive sur le marché des travailleurs. De plus, implémenter une application utilisable dans un nombre important de langues serait une bonne idée, les utilisateurs d'applications business ne se limitant pas à un seul territoire. Enfin, ces utilisateurs, étant constamment en mouvement, apprécieraient d'avoir une application comme celle-ci, à laquelle ils auraient accès depuis leur iWatch.

5.3.15 Météo

Le modèle implémenté dans le cadre des applications dont le genre principal est "Weather" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 100.

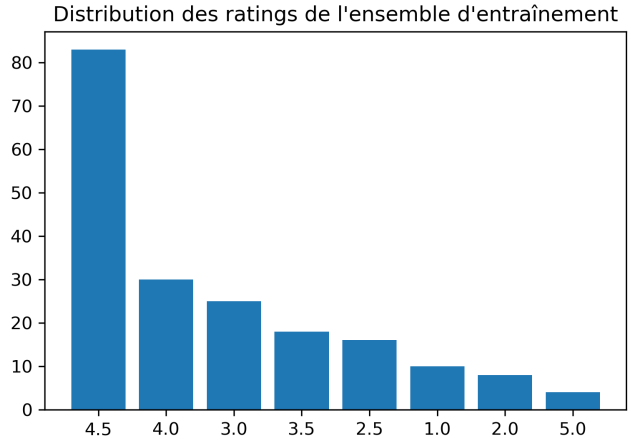

Figure 21 : Distribution des applications Météo

De manière globale, le score de classification pour les données de test est de 0.37502 et le nombre de caractéristiques utilisé est de 104. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant la classe 4.5 au lieu des classes réelles, ce qui fait sens étant donné que c'est la classe avec le plus de données dans l'ensemble de données d'entraînement (Figure 21). La matrice montre également que le modèle ne parvient pas à prédire les données de la classe 5 ni la majorité des données de la classe 4, ce qui fait sens étant donné que la classe 5 est celle avec le moins de données d'entraînement et la classe 4 a également une faible quantité de données d'entrainement comparé à la classe 4.5 bien que ce soit la deuxième classe la plus peuplée.

Features	Importance
supportedDevices_'Watch4-Watch4'	0.045694
sellerUrl_sellerUrl	0.044810
genres_'News'	0.039308
releaseDate_9	0.029892
fileSizeClassMB_Big	0.029729
NbrlanguageCodesISO2A<5	0.026856
releaseDate_12	0.026786
releaseDate_8	0.026110
fileSizeClassMB_Small	0.023956
releaseDate_2	0.023288

Table 16 : Classement des caractéristiques les plus importantes de la catégorie Météo

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 16. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type news pour compléter la météo d'informations en lien avec celle-ci (canicules, inondations, etc). Ajouter à cela, le fait de pouvoir accéder à l'application depuis son iWatch à tout moment rendrait l'application encore plus complète. De plus, implémenter une application de petite taille est plutôt attractif pour ce type d'application car les consommateurs préfèrent utiliser leur espace de stockage pour leurs photos, leur musique ou encore leur boîte e-mail. Enfin, il semblerait que les mois optimaux pour sortir ce type d'application soient les mois où il fait très chaud c'est-à-dire en août et en septembre, et, à l'inverse, ceux où il fait très froid c'est-à-dire décembre et février.

5.3.16 Stickers

Le modèle implémenté dans le cadre des applications dont le genre principal est "Stickers" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 75.

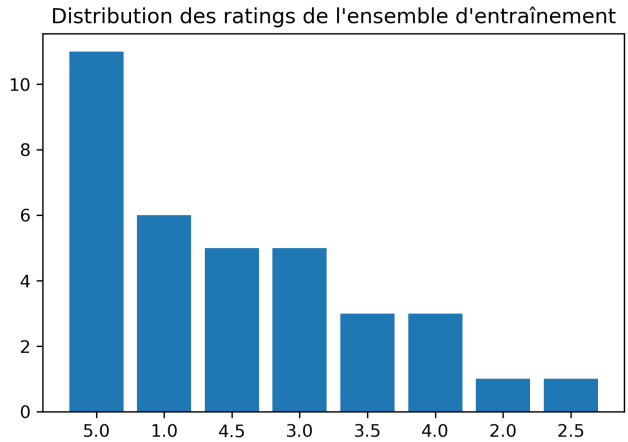

Figure 22 : Distribution des applications Stickers

De manière globale, le score de classification pour les données de test est de 0.2113 et le nombre de caractéristiques utilisé est de 64. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant la classe 5 au lieu des classes réelles, ce qui fait sens étant donné que c'est la classe avec le plus de données dans l'ensemble de données d'entraînement (Figure 22). La matrice montre également que le modèle ne parvient pas à prédire les données de la classe 4, ce qui est probablement dû à la faible quantité de données d'entrainement pour cette classe.

Features	Importance
formattedPrice_$0.99	0.062827
formattedPrice_Free	0.048624
releaseNotes_releaseNotes	0.045831
releaseDate_9	0.042680
formattedPrice_$1.99	0.035625
formattedPrice_$2.99	0.035414
sellerUrl_sellerUrl	0.027277
genres_'Gaming'	0.026859
genres_'Emoji & Expressions'	0.025348
releaseDate_8	0.024803

Table 17 : Classement des caractéristiques les plus importantes de la catégorie Stickers

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 17. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre doivent prendre en considération le fait que son prix est un facteur qui pèse énormément dans la balance dans l'évaluation de l'application. En effet, les utilisateurs d'applications partent du principe que, dès lors qu'ils paient pour une application, celle-ci doit avoir un certain niveau de qualité. Ils s'attendent à ce que leurs réclamations, rédigées dans les commentaires, soient entendues et prises en considération. Il semble dès lors opportun qu'une note soit insérée lorsqu'une mise à jour est effectuée, afin d'expliquer les dernières modifications apportées à l'application. De plus, ce genre d'application devrait inclure des fonctionnalités propres aux applications de types jeux ou emoji et expression. En effet, ce sont des catégories qui sont en lien étroit avec les stickers et qui permettent de rassembler, sur la même application, des éléments constitutifs de stickers. Enfin, il semblerait que les mois optimaux pour sortir ce type d'application soient ceux d'août et de septembre.

5.3.17 Finance

Le modèle implémenté dans le cadre des applications dont le genre principal est "Finance" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 15.

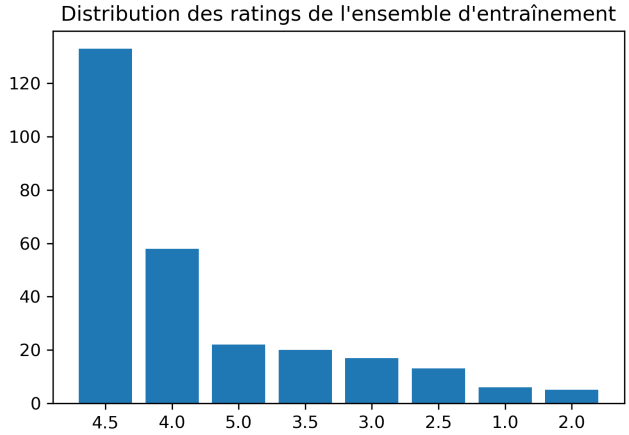

Figure 23 : Distribution des applications Finance

De manière globale, le score de classification pour les données de test est de 0.34484 et le nombre de caractéristiques utilisé est de 114. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification entre les classes 4 et 4.5, ce qui fait sens étant donné que ce sont les classes avec le plus de données dans l'ensemble de données d'entraînement (Figure 23). Elle montre également que le modèle parvient difficilement à prédire efficacement la classe 5, ce qui peut être expliqué par l'ensemble de données de la classe qui est comparativement moins important que celui des classes 4 et 4.5.

Features	Importance
sellerUrl_sellerUrl	0.047508
genres_'Business'	0.041641
genres_'Utilities'	0.040081
fileSizeClassMB_Small	0.036201
fileSizeClassMB_Big	0.030948
releaseDate_11	0.030875
supportedDevices_'Watch4-Watch4'	0.030298
features_nan	0.027098
releaseDate_2	0.026427
fileSizeClassMB_Medium	0.024150

Table 18 : Classement des caractéristiques les plus importantes de la catégorie Finance

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 18. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types business ou utilitaires. En effet, ce sont des catégories qui sont en lien étroit avec le thème de la finance et qui permettent d'optimiser une application de ce genre. De plus, implémenter une application qui est utilisable sur les iWatchs est important pour que les consommateurs puissent avoir accès à l'application à tout moment étant donné que ceux-ci sont souvent très occupés ou en mouvement. Enfin, il

semblerait que les mois optimaux pour sortir ce type d'application soient ceux de novembre et février.

5.3.18 Nourriture & Boissons

Le modèle implémenté dans le cadre des applications dont le genre principal est "Food & Drink" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 75.

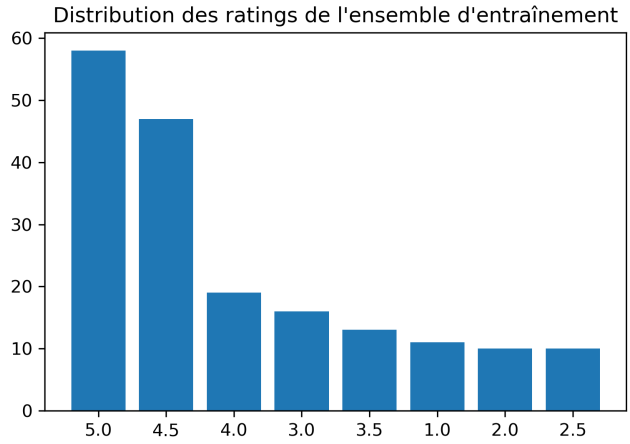

Figure 24 : Distribution des applications Nourriture & Boissons

De manière globale, le score de classification pour les données de test est de 0.27619 et le nombre de caractéristiques utilisé est de 94. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classes 4.5 et 5 au lieu des classes réelles, ce qui fait sens étant donné que ces classes sont celles avec le plus de données dans l'ensemble de données d'entraînement (Figure 24). Cependant, la matrice montre également que le modèle a du mal à prédire les données des classes 4.5 et 5 et ce, bien que ce soient les classes avec le plus de données d'entraînement. Ce paradoxe peut être expliqué par le fait que la taille de l'ensemble de données d'entrainement n'est pas conséquent.

Features	Importance
genres_'Lifestyle'	0.053838
sellerUrl_sellerUrl	0.045872
fileSizeClassMB_Big	0.036365
fileSizeClassMB_Small	0.033580
releaseDate_10	0.029287
releaseDate_12	0.028924
fileSizeClassMB_Medium	0.027341
releaseNotes_releaseNotes	0.027242
releaseDate_6	0.026980
releaseDate_4	0.024334

Table 19 : Classement des caractéristiques les plus importantes de la catégorie Nourriture & Boissons

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 19. Sur base de ces caractéristiques, es entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type lifestyle car c'est une catégorie qui est en lien étroit avec le thème nourriture et boisson. Cela permettrait de rassembler, sur la même application, des éléments qui sont dans l'air du temps et pourrait satisfaire les attentes de beaucoup de consommateurs de ce type d'application. De plus, lorsqu'une application qui est dans l'ère des blogs, réseaux sociaux et autres est sortie et réussie, elle génère généralement une communauté très active. Dès lors, il faut faire attention à bien lire les feed-backs des utilisateurs, mettre-à-jour l'application sur base de ceux-ci et informer les utilisateurs des modifications opérées par le biais de notes.

5.3.19 Conceptions graphiques

Le modèle implémenté dans le cadre des applications dont le genre principal est "Graphics & Designs" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 100.

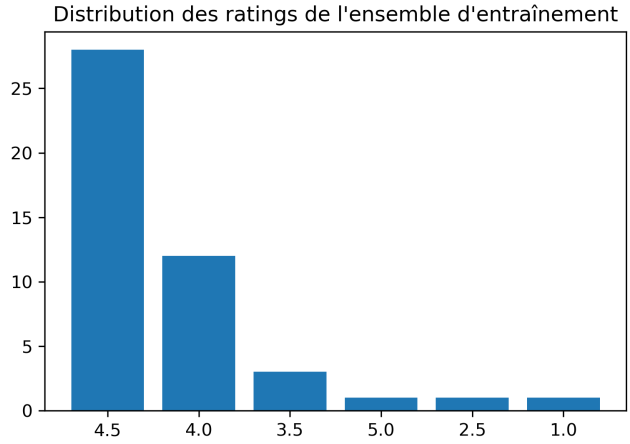

Figure 25 : Distribution des applications Conceptions graphiques

De manière globale, le score de classification pour les données de test est de 0.34137 et le nombre de caractéristiques utilisé est de 70. La matrice de confusion des données test est une matrice de dimension 4 qui ne reprend que les classes 2.5, 3, 4 et 4.5. Celle-ci montre que le modèle fait majoritairement des erreurs de classification en prédisant la classe 4.5 au lieu des classes réelles, ce qui fait sens étant donné que c'est la classe qui a le plus de données dans l'ensemble de données d'entraînement (Figure 25).

Features	Importance
sellerUrl_sellerUrl	0.062811
releaseDate_6	0.040216
genres_'Productivity'	0.037587
NbrlanguageCodesISO2A<5	0.034904
features_['iosUniversal']	0.034811
genres_'Utilities'	0.033146
releaseDate_2	0.032318
NbrlanguageCodesISO2A[5-20[	0.032285
genres_'Photo & Video'	0.032260
features_nan	0.029466

Table 20 : Classement des caractéristiques les plus importantes de la catégorie Conceptions graphiques

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 20. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type photos et vidéos, utilitaires ou encore productivité. En effet, ce sont des catégories qui sont en lien étroit avec les conceptions graphiques et qui permettent de rassembler, sur la même application, des éléments qui pourraient satisfaire les attentes de beaucoup de consommateurs de ce type d'application. De plus, implémenter une application qui est utilisable dans un nombre important de langues serait

une bonne idée puisqu'elle ne s'adresse pas à un public spécifique, géographiquement parlant. De plus, il semblerait que les mois optimaux pour sortir ce type d'application soient ceux de Juin et février. Enfin, puisqu'il s'agit d'une application employée sur le temps-libre, il serait intéressant qu'elle puisse être utilisée sur tous les types d'appareils iOS.

5.3.20 Voyages

Le modèle implémenté dans le cadre des applications dont le genre principal est "Travel" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 75.

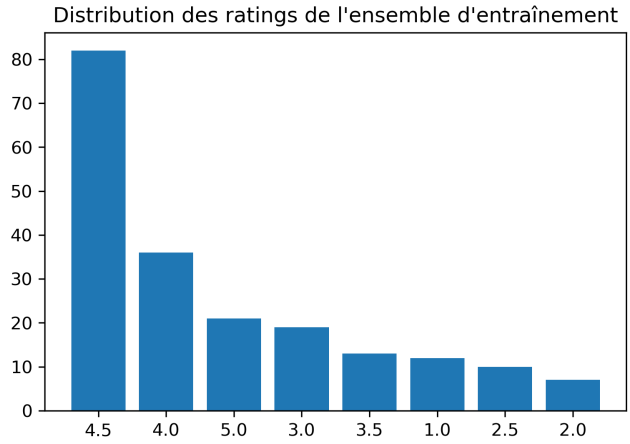

Figure 26 : Distribution des applications Voyages

De manière globale, le score de classification pour les données de test est de 0.27229 et le nombre de caractéristiques utilisé est de 104. La matrice de confusion des données test montre que le modèle fait majoritairement des erreurs de classification en prédisant les classes 4 et 4.5 au lieu des classes réelles, ce qui fait sens étant donné que ces classes sont celles avec le plus de données dans l'ensemble de données d'entraînement (Figure 26). La matrice montre également que le modèle ne parvient pas à prédire les données de la classe 5 et ce, bien que ce soit la 3eme classe avec le plus de données d'entraînement.

Features	Importance
sellerUrl_sellerUrl	0.044313
fileSizeClassMB_Big	0.033319
genres_'Lifestyle'	0.031901
genres_'Business'	0.030100
NbrlanguageCodesISO2A<5	0.029614
genres_'Navigation'	0.029370
fileSizeClassMB_Medium	0.029278
features_['iosUniversal']	0.024626
fileSizeClassMB_Small	0.024165
ipadScreenshotUrls_ipadScreenshotUrls	0.023520

Table 21 : Classement des caractéristiques les plus importantes de la catégorie Voyages

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont : sont reprises dans la table 21. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types navigation ou lifestyle. En effet, ce sont des catégories qui sont en lien étroit avec le thème voyage et qui permettent de rassembler, sur la même application, des éléments qui pourraient satisfaire les attentes de beaucoup de consommateurs de ce type d'application. De plus, implémenter une application de petite taille est plutôt attractif pour ce type d'application car les consommateurs préfèrent utiliser leur espace de stockage pour leurs photos, leur musique ou encore leur boîte e-mail. Enfin, puisqu'il s'agit d'une application employée sur le temps-libre, il serait intéressant qu'elle puisse être utilisée sur tous les types d'appareils iOS.

5.3.21 Réseaux sociaux

Le modèle implémenté dans le cadre des applications dont le genre principal est "Social Networking" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : 'True', 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : '75'.

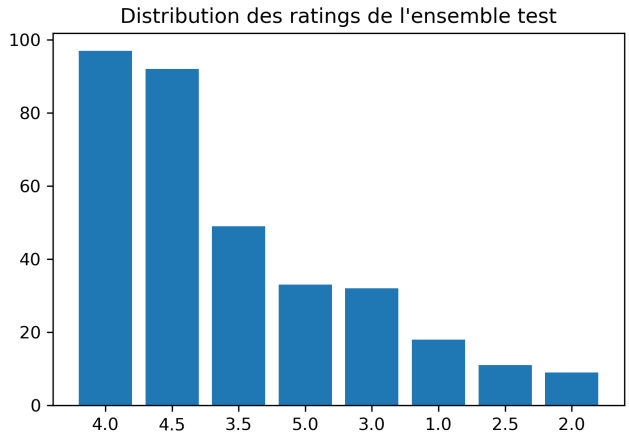

Figure 27 : Distribution des applications Réseaux Sociaux

De manière globale, le score de classification pour les données de test est de 0.29513 et le nombre de caractéristiques utilisé est de 169. La matrice de confusion des données test montre que le modèle permet essentiellement de prédire des applications à ratings élevé, ce qui fait sens puisqu'il s'agit des classes majoritaires dans cet ensemble de données (Figure 27). La majorité des erreurs de classification se fait au sein des classes 3.5, 4, 4.5 et 5, qui constituent majoritairement les classes relatives aux applications à évaluations positives.

Features	Importance
sellerUrl_sellerUrl	0.036552
genres_'Lifestyle'	0.026453
NbrlanguageCodesISO2A<5	0.025695
genres_'Entertainment'	0.025330
fileSizeClassMB_Big	0.025064
NbrlanguageCodesISO2A[5-20[	0.021981
advisories_'Infrequent/Mild Sexual Content and...	0.021860
ipadScreenshotUrls_ipadScreenshotUrls	0.020553
genres_'Utilities'	0.020185
fileSizeClassMB_Medium	0.020099

Table 22 : Classement des caractéristiques les plus importantes de la catégorie Réseaux sociaux

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 22. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types lifestyle, divertissement ou utilitaire. En effet l'utilisation d'applications de ce genre se fait généralement à des fins de divertissement. Dès lors, inclure des caractéristiques issues de ces catégories d'application apporterait de bonnes raisons aux utilisateurs de passer plus de temps sur l'application. De plus, il est préférable qu'elle soit disponible dans plus de 5 langues afin d'étendre au maximum le public cible. Enfin, prévoir une version compatible avec l'Ipad semble également important pour permettre aux utilisateurs de s'en servir avec plus de fluidité dans leurs moments de tranquillité.

5.3.22 Médical

Le modèle implémenté dans le cadre des applications dont le genre principal est "Medical" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 100.

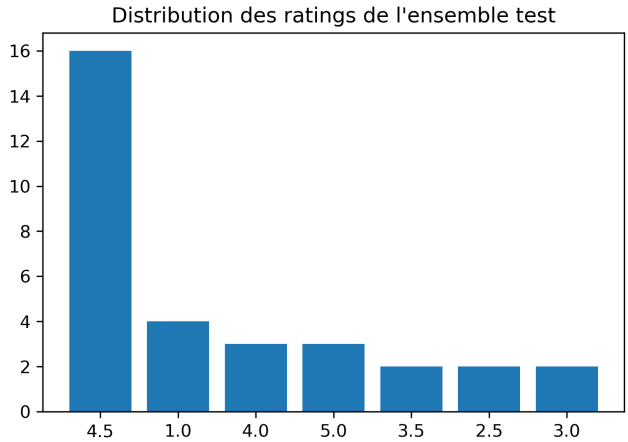

Figure 28 : Distribution des applications Médicales

De manière globale, le score de classification pour les données de test est de 0.35897 et le nombre de caractéristiques utilisés est de 87. La matrice de confusion des données test montre que le modèle fait essentiellement des erreurs en prédisant le classement dans la classe 4.5 d'applications d'autres classes ce qui fait sens étant donné que la classe 4.5 est la classe majoritaire dans l'ensemble de données d'entraînement (Figure 28).

Features	Importance
sellerUrl_sellerUrl	0.046535
genres_'Health & Fitness'	0.040755
supportedDevices_'Watch4-Watch4'	0.027915
releaseDate_3	0.024805
ipadScreenshotUrls_ipadScreenshotUrls	0.024379
releaseDate_9	0.024177
features_nan	0.023442
features_['iosUniversal']	0.022864
fileSizeClassMB_Big	0.022780
releaseDate_4	0.022637

Table 23 : Classement des caractéristiques les plus importantes de la catégorie Médicale

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 23. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de type santé et fitness, ce qui fait sens compte-tenu du fait que les 2 catégories sont fortement liées. Elles pourraient offrir un suivit médical pour les personnes faisant attention à leur santé. Également, pour ne restreindre aucun utilisateur et rendre l'application plus attractive, elles devraient la rendre compatible avec tous les appareils iOS ainsi que l'iWatch. Enfin, pour se différencier des autres et également pour les rendre plus attractives, elles pourraient proposer une application plus légère.

5.3.23 Divertissement

Le modèle implémenté dans le cadre des applications dont le genre principal est "Entertainment" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'gini', 'max_features' : 'sqrt' et 'n_estimators' : 100.

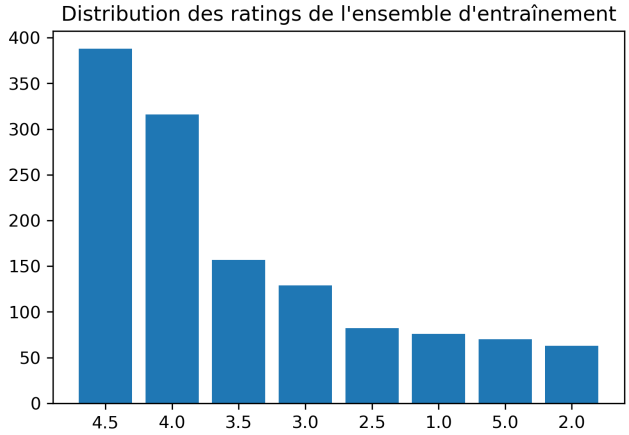

Figure 29 : Distribution des applications Divertissement

De manière globale, le score de classification pour les données de test est de 0.28291 et le nombre de caractéristiques utilisé est de 159. La matrice de confusion des données test montre que le modèle a particulièrement du mal à prédire la classe 5, ce qui fait sens étant donné qu'il s'agit d'une classe dans laquelle peu de données sont disponibles dans les données d'entraînement (Figure 29). À coté de cela, lorsqu'il se trompe, le modèle a tendance à classer les données dans les classes 4 et 4.5, ce qui fait sens car ce sont celles qui contiennent une grande partie des données d'entraînement. La majorité des erreurs de classification se fait donc au sein des classes 3.5, 4 et 4.5, qui constituent majoritairement les classes relatives aux applications bien évaluées.

Features	Importance
sellerUrl_sellerUrl	0.036884
genres_'Photo & Video'	0.026845
genres_'Lifestyle'	0.022847
releaseDate_12	0.022840
fileSizeClassMB_Big	0.022823
releaseNotes_releaseNotes	0.021743
fileSizeClassMB_Small	0.021626
genres_'Utilities'	0.021411
releaseDate_3	0.019276
NbrlanguageCodesISO2A<5	0.018501

Table 24 : Classement des caractéristiques les plus importantes de la catégorie Divertissement

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 24. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types photos et vidéos, lifestyle ou encore utilities pour pouvoir compter un nombre important de fonctionnalités qui touchent différents registres. S'agissant d'un type d'appli-

cation sujet aux mises à jour récurrentes, elle devrait être accompagnée de notes dès la sortie d'une nouvelle version. Enfin, la taille de l'application a une grande importance. Par conséquent, il faut tenter de minimiser la taille de celle-ci tout en tentant de faire en sorte que l'application fournisse des services attractifs.

5.3.24 Lifestyle

Le modèle implémenté dans le cadre des applications dont le genre principal est "Lifestyle" a été lancé sous les paramètres retournés par le GridSearchCV suivants : 'bootstrap' : True, 'criterion' : 'entropy', 'max_features' : 'sqrt' et 'n_estimators' : 25.

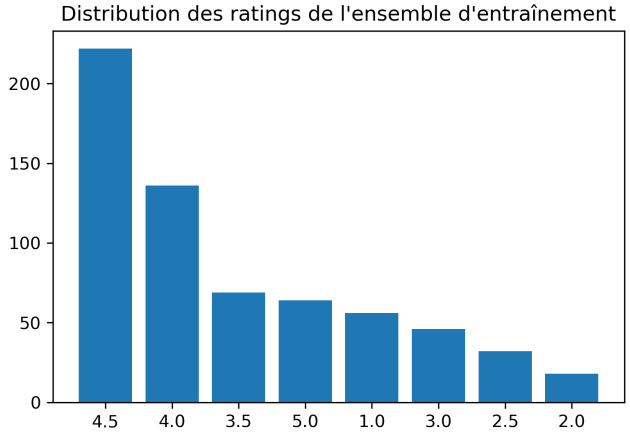

Figure 30 : Distribution des applications Lifestyle

De manière globale, le score de classification pour les données de test est de 0.26345 et le nombre de caractéristiques utilisé est de 161. La matrice de confusion des données test montre que l'essentiel des confusions de prédictions ont lieu au sein des classes 4 et 4.5, ce qui fait sens étant donné que ce sont les classes dans lesquelles il y a le plus de données d'entrainement pour cette catégorie (Figure 30). Elle montre également que le modèle a du mal à prédire la classe 5 efficacement ce qui implique qu'il n'a pas réussit à trouver les déterminants de cette classe.

Features	Importance
sellerUrl_sellerUrl	0.041085
genres_'Entertainment'	0.033324
fileSizeClassMB_Big	0.030371
genres_'Utilities'	0.027343
fileSizeClassMB_Small	0.027211
fileSizeClassMB_Medium	0.026013
NbrlanguageCodesISO2A<5	0.025443
ipadScreenshotUrls_ipadScreenshotUrls	0.022635
genres_'Social Networking'	0.022215
releaseNotes_releaseNotes	0.021516

Table 25 : Classement des caractéristiques les plus importantes de la catégorie Lifestyle

Les 10 caractéristiques qui influencent principalement la variance du rating pour cette catégorie d'application sont reprises dans la table 25. Sur base de ces caractéristiques, les entreprises qui souhaitent lancer une application de ce genre devraient inclure des fonctionnalités propres aux applications de types divertissement, ou réseaux sociaux. Ce type d'application est généralement utilisé lors des temps-libre ce qui induit l'intérêt de l'ajout de caractéristiques relatives à des applications qui sont également utilisées durant le temps libre. Elle devrait également être accompagnée de notes dès la sortie d'une nouvelle version compte-tenu de la vitesse de mise à jour de ce type de données. Enfin, elle devrait être disponible sur Ipad pour pouvoir être utilisée avec fluidité dans des moments de tranquillité.

5.3.25 Analyse inter-catégories

Les résultats des classifications par catégories ont mis en évidence un pattern qui était presque systématiquement présent : l'incapacité des modèles construits à trouver les déterminants de la classe 5. De manière générale, les modèles ne parviennent pas, ou très difficilement, à prédire des classifications d'observations dans la classe 5 justes. Les observations qui se classent dans cette catégorie sont généralement peu nombreuses, ce qui pourrait expliquer la raison de ces erreurs de classification. Cependant, les classes à ratings faibles sont aussi à quantité peu élevée mais mènent à moins d'erreurs que pour la classe 5. Dès lors, ce pattern pousse à se demander si cette classe est difficile à prédire correctement à cause du manque de données ou si c'est pour une raison autre.

Les caractéristiques principales recensées dans les différentes catégories sont rassemblées dans la Table 26 et sont ordonnées par nombre d'occurrence. Après l'analyse par catégorie, une analyse inter-catégorie est nécessaire pour reprendre les dimensions les plus récurrentes dans les caractéristiques les plus importantes.

Caractéristiques	Nombre d'occurrence	Caractéristiques	Nombre d'occurrence
sellerUrl_sellerUrl	24	advisories_'Infrequent/Mild Sexual Content and Nudity'	2
fileSizeClassMB_Big	20	genres_'Family'	2
fileSizeClassMB_Small	19	genres_'Education'	2
NbrlanguageCodesISO2A<5	15	genres_'Photo & Video'	2
supportedDevices_'Watch4-Watch4'	12	genres_'Gaming'	1
fileSizeClassMB_Medium	11	minimumOsVersion_9	1
genres_'Utilities'	10	releaseDate_5	1
genres_'Entertainment'	9	genres_'News'	1
genres_'Lifestyle'	8	genres_'Casual'	1
releaseNotes_releaseNotes	7	genres_'Social Networking'	1
NbrlanguageCodesISO2A[5-20[	7	genres_'Adventure'	1
releaseDate_9	6	genres_'Health & Fitness'	1
releaseDate_2	6	genres_'Travel'	1
ipadScreenshotUrls_ipadScreenshotUrls	5	genres_'Medical'	1
releaseDate_7	5	formattedPrice_$1.99	1
features_nan	5	genres_'Emoji & Expressions'	1
releaseDate_8	5	genres_'Games'	1
releaseDate_12	5	formattedPrice_$0.99	1
releaseDate_4	4	genres_'Simulation'	1
releaseDate_3	4	formattedPrice_$2.99	1
features_['iosUniversal']	4	formattedPrice_Free	1
releaseDate_11	3	genres_'Action'	1
genres_'Business'	3	genres_'Navigation'	1
releaseDate_6	3	genres_'Puzzle'	1
genres_'Productivity'	3	genres_'Weather'	1
minimumOsVersion_8	2	releaseDate_1	1
supportedDevices_'Watch5-Watch5'	2	advisories_'Infrequent/Mild Realistic Violence'	1
releaseDate_10	2		

Table 26 : Classification des caractéristiques les plus récurrentes

Il est intéressant de noter que la caractéristique qui est le plus souvent répertoriée est la présence de l'URL de l'entreprise fournissant l'application. Cet élément correspond au résultat obtenu lors de l'analyse globale. Dès lors, pour tout créateur d'application, il est primordial pour lui d'inclure un moyen de consulter ses données pour pouvoir rassurer les utilisateurs.

La taille de l'application semble également être un aspect important pour les utilisateurs. Il faut dès-lors privilégier les applications de petite taille et ne concevoir des applications lourdes que lorsqu'il y a une réelle plus-value sous-jacente pour l'utilisateur.

Un autre aspect qui semble énormément importer est la possibilité d'utiliser l'application sur iWatch. Compte-tenu du nombre d'occurrence de cette caractéristique dans les différentes catégories et de la vitesse de croissance des nouvelles technologies, il est très probable que cette caractéristique devienne un incontournable pour toute bonne application dans le futur.

Les genres dans lesquels l'application peut être catégorisée constitue également un aspect important. Comme mentionné dans l'analyse globale, les applications de types divertissement constituent un marché très intéresssant pour tout lanceur d'application. Les applications de type utilitaire et lifestyle sont également des applications en vue sur le marché des applications. Les utilitaires sont là pour faciliter la vie des utilisateurs et les applications lifestyle sont dans l'ère des réseaux-sociaux, des bloggeurs, des voyages documentés, etc. Ces applications sont également créées pour faciliter la vie des utilisateurs et englobent un nombre important de sous-catégories, d'où son intérêt.

L'ajout de notes lorsque de nouvelles versions de l'application sortent fait aussi partie des caractéristiques les plus réccurrentes dans les différents types d'application. Cette caractéristique est importante dans le sens où elle permet de montrer aux utilisateurs que leurs réclamations sont entendues et permet également de maintenir une certaine relation avec eux en les informant dès qu'il y a quelque chose de nouveau dans l'application.

L'aspect temporel relatif à la sortie de l'application a l'air de jouer un rôle conséquent sur son succès. En effet, au vu du nombre de fois que les mois de sortie des applications constituent certains des facteurs qui affectent fortement la variance du rating, ça pousse à la réflexion quant à son importance. Dès lors, compte-tenu de la catégorie et des fonctionnalités de l'application, il est important de bien choisir le moment de lancement de celle-ci car il peut influencer énormément le succès potentiel de l'application.

Enfin, il est important de mentionner le manque de présence du prix dans les caractéristiques les plus importantes. Ce point montre que l'hypothèse de [5] selon laquelle, lorsqu'une application est de qualité, il n'est pas problématique pour les utilisateurs d'y mettre le prix n'est pas erronée. Selon cette étude, le prix d'une application, aussi élevé soit-il, ne fait que rarement partie des facteurs qui influencent le plus la variance du rating de l'application.

6 Discussion

Au vu de la vitesse de croissance du marché des applications mobiles, il devient de plus en plus difficile pour les offreurs sur ce marché de se différentier. Les analyses précédemment réalisées reprenant les feed-backs des utilisateurs et rassemblant les critiques les plus couramment émises constituent une bonne méthode pour arriver à contenter les utilisateurs. En effet, il est important de miser sur les aspects esthétiques et qualitatifs de l'application pour qu'elle soit appréciée des utilisateurs. Cependant, offrir de nouveaux services, des services innovants ou encore des services parapluie qui en rassemblent plusieurs peut également constituer une manière de percer dans un marché aussi concurrentiel que celui-ci. Dès lors, prendre en considération des aspects techniques, qui ne seront pas forcément mentionnés dans ces analyses de feed-back, et anticiper les attentes des consommateurs en leur fournissant un service unique est également important. Cette étude avait pour objet de définir les critères techniques de succès d'une application dans l'App Store et a permis de définir des moyens pour arriver à potentiellement atteindre ce succès. Cette étude met en évidence ces critères de succès et les formule sous forme de recommandations à suivre pour potentialiser les chances qu'a l'application d'arriver à un bon rating. Dès lors, en tant que créateur d'application, suivre ces recommandations ne permettrait probablement pas à l'application d'aboutir à un rating maximal immédiat mais permettrait d'augmenter ses chances d'y arriver.

Catégories	F1 Score	Catégories	F1 Score
Modèle général	32,41%	Utilities	26,33%
Social Networking	29,51%	Education	23,44%
Medical	35,9%	Health & Fitness	38,01%
Entertainment	28,29%	Sports	30,72%
Lifestyle	26,34%	News	34,01%
Navigation	15,51%	Business	23,99%
Games	34,48%	Weather	37,50%
Reference	32,12%	Stickers	21,13%
Photo & Video	30,4%	Finance	34,48%
Book	29,26%	Food & Drink	27,62%
Music	28,81%	Graphics & Design	34,14%
Productivity	25,2%	Travel	27,23%
Shopping	35,01%		

Table 27 : Scores de classification par catégorie d'application

Les recommandations rassemblées dans cette étude reposent essentiellement sur des hypothèses issues des caractéristiques influençant le plus la variance. Ces caractéristiques s'avèrent être beaucoup plus représentatives pour certaines catégories que d'autres au vu de la taille des ensembles de données considérés dans certains cas ou des scores de classification dans d'autres. Pour chacun des modèles construits, le score de classification a dépassé le benchmark initialement établi à 0.125 (Table 27), ce qui implique que l'utilisation de ces modèles constitue, potentiellement, une bonne première piste pour pouvoir aboutir à des prédictions sur la classification d'applications dans leur rating correspondant.

7 Limites

Les résultats obtenus lors de cette étude exploratoire doivent être mis en perspective et être pris avec du recul sachant que les recommandations énoncées sont basées sur un grand nombre d'hypothèses. Ils ne peuvent pas être considérés comme des règles de décision qui mèneront directement à un rating élevé. En effet, bien que le benchmark initialement établi a été dépassé dans chacune des catégorie analysée, il y a de nombreux facteurs qui ont entravé l'aboutissement à des résultats optimaux.

Premièrement, la taille de l'ensemble de données recensé a limité l'arrivée à des résultats représentatifs car, lorsqu'un modèle est entraîné sur des ensembles de données de petite taille, il a tendance à ne pas avoir les tenants, les déterminants et les aboutissants ce qui mène à des résultats avec des scores faibles.

Ensuite, au vu des scores des modèles implémentés observés, il est clair qu'il manquait, des facteurs décisifs pour pouvoir aboutir à de bonnes prédictions. Compte-tenu des élément recueillis dans la revue de littérature, il y a un grand nombre de facteurs qualitatifs qui n'ont pas été repris dans les caractéristiques de l'ensemble de données qui sont très pertinents.

De plus, les données relatives aux ratings évalués à 5 posent un grand nombre d'interrogations. En effet, une application ayant un rating de 5 devrait constituer une application "parfaite" qui satisferait toutes les attentes de l'individu l'ayant évaluée, ce qui est très difficilement envisageable. Il y a donc un risque que ces applications aient fait l'objet d'une évaluation biaisée. Dans l'étude réalisée, il y a très peu de cas où les classifications dans cette classe s'est avérée juste. C'est pourquoi il faudrait investiguer cette catégorie et s'assurer qu'elle ne compte pas un grand nombre d'évaluations biaisées. Car, si tel est le cas, cela rend également les modèles de prédiction définis biaisés.

Enfin, certaines caractéristiques qui fournissaient très probablement beaucoup d'informations ont dû être abandonnées par soucis de temps. En effet, les caractéristiques « description » et « releaseNotes » auraient nécessité un travail plus approfondi pour permettre de dégager des caractéristiques traitables dans le cadre de ce travail.

Dès lors, dans le cadre d'une seconde version de cette étude, un ensemble de données plus conséquent devrait être établi en vue d'aboutir à des résultats plus représentatifs. De plus, un traitement des caractéristiques composées de textes continus par text mining pour dégager des sous- caractéristiques pourrait permettre l'amélioration des résultats de classification. Enfin, la jointure de caractéristiques qualitatives à cette analyse exploratoire devrait permettre de retourner de meilleurs résultats et, par conséquent, un meilleur score. Par conséquent, cette seconde version permettrait de confirmer ou infirmer la validité des hypothèses posées.

8 Conclusion

La question de recherche dans le cadre de cette étude était la suivante : « Quels sont les critères de succès d'une application dans l'App Store ». Pour parvenir à y répondre, cette étude exploratoire en plusieurs étapes a été entreprise.

Dans un premier temps, une revue de littérature a été établie pour pouvoir s'informer de ce qui avait déjà été fait en la matière. Il s'est avéré qu'une grande partie des études réalisées par le passé ont été faites sur base des feed-backs et des ratings des applications en vue de comprendre les raisons de mécontentement des utilisateurs d'applications. Mais la recherche d'une relation entre les données techniques publiées sur le magasin d'application et les ratings elle, n'avait pas encore été entreprise.

Dans un second temps, un recueil des données pour pouvoir réaliser les analyses a eu lieu par le biais d'un script écrit en python. Ce recueil a utilisé 2 méthodes qui sont le « lookup » et le « search », la première étant très lente, elle dut laisser place à la seconde par mesure de facilité. Ensuite un nettoyage des données a été entrepris, suivi d'un traitement des caractéristiques pour aboutir à l'ensemble de données final.

Ensuite, la mise en place de modèles de prédictions sous forme de forêts aléatoires a été réalisée pour pouvoir récupérer les caractéristiques faisant le plus varier le rating des applications.

Enfin, une analyse des résultats a eu lieu en posant des hypothèses pour pouvoir donner des recommandations aux managers des entreprises souhaitant lancer leur application, peu importe le type.

En conclusion, il est possible qu'il existe des liens entre les critères techniques des applications et la variation de leur rating compte tenu du fait qu'avec les modèles de prédiction développés, les résultats sont meilleurs que le benchmark établi de 0.125. Cependant une étude supplémentaire mêlant le qualitatif et le quantitatif permettra de réellement répondre à cette question.

Bibliographie

[1] itunes search api. `https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api/`.

[2] The most popular app store keywords of 2020. `https://www.apptweak.com/aso-blog/the-most-popular-app-store-keywords`.

[3] Kishore Baktha. Mobile application development : All the steps and guidelines for successful creation of mobile app : Case study. *International Journal of Computer Science and Mobile Computing*, 6(9) :15–20, 2017.

[4] Timothy Bresnahan, Joe Orsini, and Pai ling Yin. Platform choice by mobile app developers. 2014.

[5] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. Why people hate your app : Making sense of user feedback in a mobile app store. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 1276–1284, 2013.

[6] Zena Hira and Duncan Gillies. A review of feature selection and feature extraction methods applied on microarray data. *Advances in Bioinformatics*, 2015 :1–13, 2015.

[7] Sami Hyrynsalmi, Tuomas Mäkilä, Antero Järvi, Arho Suominen, Marko Seppänen, and Timo Knuutila. App store, marketplace, play ! an analysis of multi-homing in mobile software ecosystems. In *Proceedings of the International Workshop on Software Ecosystems (IWSECO 2012)*, page 59–72, 2012.

[8] Andrei Idu, Tommy Zande, and Slinger Jansen. Multi-homing in the apple ecosystem : Why and how developers target multiple apple app stores. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems (MEDES'11)*, pages 122–128, 2011.

[9] Dr. MD Rashedul Islam and Tridib Mazumder. Mobile application and its global impact. *International Journal of Engineering Technology*, 10(6) :72–78, 2010.

[10] He Jiang, Hongjing Ma, Zhilei Ren, Jingxuan Zhang, and Xiaochen Li. What makes a good app description ? In *Proceedings of the 6th Asia-Pacific Symposium on Internetware on Internetware*, INTERNETWARE 2014, page 45–53, New York, NY, USA, 2014.

[11] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. *2014 Science and Information Conference*, pages 372–378, 2014.

[12] Jieun Kim, Yongtae Park, Chulhyun Kim, and Hakyeon Lee. Mobile application service networks : Apple's app store. *Service Business*, 8 :1–27, 2014.

[13] W. Martin. Causal impact for app store analysis. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 659–661, 2016.

[14] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A survey of app store analysis for software engineering. *IEEE Transactions on Software Engineering*, 43(9) :817–847, 2017.

[15] I. J. Mojica Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan. Examining the rating system used in mobile-app stores. *IEEE Software*, 33(6) :86–92, 2016.

[16] D. Pagano and W. Maalej. User feedback in the appstore : An empirical study. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 125–134, 2013.

[17] Mahesh Pal. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing - INT J REMOTE SENS*, 26 :217–222, 2005.

[18] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn : Machine learning in python. *Journal of Machine Learning Research*, 12(85) :2825–2830, 2011.

[19] Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. App recommendation : A contest between satisfaction and temptation. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, page 395–404, 2013.

Appendices

Annexe A

Descriptions des caractéristiques

Caractéristique	Explication
advisories	Publicités attribuées par Apple
amgArtistId	ID AMG de l'auteur (All Media Guide)
appletvScreenshotUrls	URLs des captures d'écran Apple tv
artistId	ID de l'auteur
artistLinkUrl	URL de l'auteur
artistName	Nom de l'auteur
artistType	Type d'auteur
artistViewUrl	URL de l'auteur sur l'App Store
artworkUrl100	URL de l'œuvre en 100 * 100 pixels
artworkUrl30	URL de l'œuvre en 30 * 30 pixels
artworkUrl512	URL de l'œuvre en 512 * 512 pixels
artworkUrl60	URL de l'œuvre en 60 * 60 pixels
artworkUrl600	URL de l'œuvre en 600 * 600 pixels
averageUserRating	Moyenne des évaluations des utilisateurs
averageUserRatingForCurrentVersion	Moyenne des évaluations des utilisateurs pour la version actuelle
bundleId	ID unique de l'application attribué par le développeur
collectionArtistId	ID de l'auteur de l' album, la saison,…
collectionArtistName	Nom de l'auteur de l' album, la saison,…
collectionArtistViewUrl	URL de l'auteur de l' album, la saison,… sur l'App Store
collectionCensoredName	Nom censuré de l' album, la saison,…
collectionExplicitness	Niveau d'explicité du contenu de l' album, la saison,…
collectionHdPrice	Prix de l' album, la saison,… en haute-définition
collectionId	ID de l' album, la saison,…
collectionName	Nom de l' album, la saison,…
collectionPrice	Prix de l' album, la saison,…
collectionType	Type de l' album, la saison,…
collectionViewUrl	URL de l' album, la saison,… sur l'App Store
contentAdvisoryRating	Rating du contenu par le conseiller de contenu
copyright	Copyright
country	Pays
currency	Devise
currentVersionReleaseDate	Date de sortie de la version courante
description	Description du contenu
discCount	Nombre de disques
discNumber	Numéro de disque
features	Caractéristiques
feedUrl	URL du flux RSS (Really Simple Syndication)
fileSizeBytes	Taille du fichier en octets
formattedPrice	Prix au format de la devise

genreIds	IDs des catégories dont fait partie le contenu
genres	Catégories dont fait partie le contenu
index	/
ipadScreenshotUrls	URLs des captures d'écran iPad
isGameCenterEnabled	Si le Game Center est activé pour le contenu
isStreamable	Si le contenu est disponible sur Apple Music
isVpp- DeviceBasedLicensingEnabled	Si l'application suppporte la distribution VPP (Programme d'achat en volume)
kind	Genre du contenu
languageCodesISO2A	Codes de langue ISO2A supportés
longDescription	Longue version de la description
minimumOsVersion	Version OS minimum requise pour l'utilisation du contenu
previewUrl	URL vers un aperçu des 30 premières secondes du contenu
price	Prix
primaryGenreId	ID de la catégorie principale
primaryGenreName	Nom de la catégorie principale
releaseDate	Date de sortie
releaseNotes	Notes de la version
screenshotUrls	URLs des captures d'écran
sellerName	Nom du Vendeur
sellerUrl	URL du vendeur
shortDescription	Courte version de la description
supportedDevices	Appareils compatibles
trackCensoredName	Nom censuré du contenu
trackContentRating	Rating du contenu
trackCount	Nombre de pistes du contenu
trackExplicitness	Niveau d'explicité du contenu
trackHdPrice	Prix de la version haute-définition du contenu
trackHdRentalPrice	Prix de location de la version haute-définition du contenu
trackId	ID unique du contenu attribué par Apple
trackName	Nom du contenu
trackNumber	Numéro du contenu
trackPrice	Prix du contenu
trackRentalPrice	Prix de location du contenu
trackTimeMillis	Durée du contenu en millisecondes
trackViewUrl	URL du contenu sur l'App Store
userRatingCount	Nombre total de ratings d'utilisateurs
userRatingCountForCurrentVersion	Nombre de ratings d'utilisateurs pour la version actuelle
version	Numéro de version
wrapperType	Nom de l'objet

Annexe B

Script en Python

Thesis - Umuhoza Patience

August 17, 2020

1 Critères techniques de succès d'une application dans l'App Store

[ ]:
```
import urllib3
import requests
import json
import numpy as np
import time
import pandas as pd
#pd.set_option('display.max_rows', 10)


urllib3.disable_warnings()

#CHANGE SERVER LIMIT :
#jupyter notebook --NotebookApp.iopub_data_rate_limit=10000000000
```

1.1 Récupération de données

[ ]:
```
#Lookup method
url = 'https://itunes.apple.com/lookup?'
# x = ID value
x = 500833385
# y = Array containing request result
y = []
#counter = counter of results entries -> same as len(y)
counter = 0
for i in range (100000000):
#    while True:
    try:
        param = { 'id': x, 'country':'us', 'entity':'software', 'limit': '200'}
        res = requests.get(url, param, verify=False)
        res.raise_for_status()
        res = res.json()
        if (res['results']):
            print('OK', i)
            for j in range(len(res['results'])):
                y.append(res['results'][j])
```

```
                counter += 1
        else:
            print('NO', i)
        x += 1
        time.sleep(2)

#Put "continue" after the print statement so that the printing actually gets␣
 ↪done and the system can keep on looping
    except requests.exceptions.HTTPError as errh:
        x += 1
        print ("Http Error:",errh)
        continue
    except requests.exceptions.ConnectionError as errc:
        x += 1
        print ("Error Connecting:",errc)
        continue
    except requests.exceptions.Timeout as errt:
        x += 1
        print ("Timeout Error:",errt)
        continue
    except requests.exceptions.RequestException as err:
        x += 1
        print ("OOps: Something Else",err)
        continue
```

[ ]:
```
#Save Lookup data
with open('/Users/patienceumuhoza/Documents/Mémoire/DonnéesAppStore/
 ↪DatasetThesis1.txt.js', 'w') as f:
    json.dump(y,f)
```

[ ]:
```
#Retrieve ASO 500 top keywords in 2019
FileASO = pd.read_excel(r'/Users/patienceumuhoza/Documents/Mémoire/
 ↪TOPKeywords2019.xlsx', sheet_name='US')
DataASO = pd.DataFrame(FileASO, columns= ['Keyword'])
#Keep solely the first 250 keywords
DataASO = DataASO[:250]
#DataASO = np.array(DataASO)
DataASO = list(DataASO['Keyword'])
print(DataASO)
```

[ ]:
```
#Search method
url = 'https://itunes.apple.com/search?'
# y = Array containing request result
y = []
#counter = counter of results entries -> same as len(y)
counter = 0
for i in range (len(DataASO)):
```

```
#    while True:
    try:
        param = { 'term': DataASO[i], 'country':'us', 'entity':'software',␣
 ↪'limit': '200'}
        res = requests.get(url, param, verify=False)
        res.raise_for_status()
        res = res.json()
        if (res['results']):
            print('OK', i)
            for j in range(len(res['results'])):
                y.append(res['results'][j])
                counter += 1
        else:
            print('NO', i)

        time.sleep(2)

#Put "continue" after the print statement so that the printing actually gets␣
 ↪done and the system can keep on looping
    except requests.exceptions.HTTPError as errh:
        print ("Http Error:",errh)
        continue
    except requests.exceptions.ConnectionError as errc:
        print ("Error Connecting:",errc)
        continue
    except requests.exceptions.Timeout as errt:
        print ("Timeout Error:",errt)
        continue
    except requests.exceptions.RequestException as err:
        print ("OOps: Something Else",err)
        continue
```

[ ]:
```
#Save Search data
with open('/Users/patienceumuhoza/Documents/Mémoire/DonnéesAppStore/
 ↪DatasetThesis2.txt.js', 'w') as f:
    json.dump(y,f)
```

[ ]:
```
#Assemble all the datasets built

tentative1 = pd.read_json('/Users/patienceumuhoza/Documents/Mémoire/
 ↪DonnéesAppStore/DatasetThesis1.txt.js', orient='values')
tentative2 = pd.read_json('/Users/patienceumuhoza/Documents/Mémoire/
 ↪DonnéesAppStore/DatasetThesis2.txt.js', orient='values')
dtentative1 = pd.DataFrame(tentative1)
dtentative2 = pd.DataFrame(tentative2)
```

```
dDatasetThesis = pd.concat([dtentative1, dtentative2], ignore_index=True)
dDatasetThesis = dDatasetThesis[dDatasetThesis['wrapperType'] == 'software']
dDatasetThesis = dDatasetThesis.reset_index(drop=True)

print('Length of the lookup dataset:', len(dtentative1))
print('Length of the search dataset:', len(dtentative2))
print('Length of the final dataset:', len(dDatasetThesis))
print(dDatasetThesis)

dDatasetThesis.to_json(r'/Users/patienceumuhoza/Documents/Mémoire/
 ↪DonnéesAppStore/dDatasetThesis3.txt.js', orient='values')
dDatasetThesis.to_csv(r'/Users/patienceumuhoza/Documents/Mémoire/DonnéesAppStore/
 ↪dDatasetThesis3.csv', index = False, header=True)
```

Length of the lookup dataset: 768

Length of the search dataset: 37947

Length of the final dataset: 38715

[ ]:
```
#Open dataset JSON version
with open('/Users/patienceumuhoza/Documents/Mémoire/DonnéesAppStore/
 ↪dDatasetThesis3.txt.js', 'r') as f:
    DatasetThesis = json.load(f)
    DatasetThesis = pd.DataFrame(DatasetThesis)
```

[ ]:
```
#Open dataset CSV version
dDatasetThesis = pd.read_csv('/Users/patienceumuhoza/Documents/Mémoire/
 ↪DonnéesAppStore/dDatasetThesis3.csv')
print(dDatasetThesis)
```

[ ]:
```
#Sort dataset based on artistID and remove duplicates

dFinalDatasetThesis = dDatasetThesis.drop_duplicates(subset ="trackId")
dFinalDatasetThesis = dFinalDatasetThesis.sort_values("trackName")

#Delete app data where no one gave any ratings
dFinalDatasetThesis = dFinalDatasetThesis[dFinalDatasetThesis['userRatingCount']␣
 ↪> 0]
dFinalDatasetThesis = dFinalDatasetThesis.reset_index(drop=True)
print(dFinalDatasetThesis)

dFinalDatasetThesis.to_json(r'/Users/patienceumuhoza/Documents/Mémoire/
 ↪DonnéesAppStore/dFinalDatasetThesis.txt.js', orient='values')
dFinalDatasetThesis.to_csv(r'/Users/patienceumuhoza/Documents/Mémoire/
 ↪DonnéesAppStore/dFinalDatasetThesis.csv', index = False, header=True)
```

[ ]:
```
#Open final Dataset
dFinalDatasetThesis = pd.read_csv('/Users/patienceumuhoza/Documents/Mémoire/
 ↪DonnéesAppStore/dFinalDatasetThesis.csv')
print(dFinalDatasetThesis)
#Retrieve available features
nameFeatures = list(dFinalDatasetThesis.columns)
print(nameFeatures)
```

1.2 Traitement des données

[ ]:
```
#Features handling


#Based on info() -> delete features with 0 non-null data bc they don't bring␣
 ↪anything to the table (45 features left)
print(dFinalDatasetThesis.info())
dfCleaned = dFinalDatasetThesis.dropna(axis=1, how='all')


#Based on nunique() -> delete features where all instances have the same value␣
 ↪(40 features left)
print(dfCleaned.nunique(dropna=False))
print(dfCleaned.nunique() == 1)
dfCleaned = dfCleaned.loc[:, dfCleaned.nunique() > 1]


# -> Also delete features where all instances have different values value (34␣
 ↪features left)
#BundleID (make research), artworkUrl100, artworkUrl512, artworkUrl60, trackId,␣
 ↪trackViewUrl
print(dfCleaned.nunique() == 22545)
dfCleaned = dfCleaned.loc[:, dfCleaned.nunique() != 22545]


#Use info() to change the features types -> 2 build insights on contents
print(dfCleaned.info())
#Advisories -> change type to categorical
dfCleaned['advisories'] = pd.Categorical(dfCleaned['advisories'])
#Create dummies with categories -> 2 check again -> INITIAL COLUMN 2 DROP IN␣
 ↪FINAL DUMMY DATA
adv = dfCleaned['advisories']
advClean = [x.strip('[]').split(',') for x in adv]

catega = []
for a in advClean:
    catega.append(a)
```

```
catega = pd.DataFrame(catega).stack()
catega = catega.str.strip()
advCateg = catega.unique()
advCateg = np.sort(advCateg)

advisoriesDFd = pd.DataFrame(index=dfCleaned.index, columns=advCateg)
for i in range (len(adv)):
    for j in range(len(advCateg)):
        if (advCateg[j] in adv[i]):
            advisoriesDFd.iloc[i][j] = 1
        else:
            advisoriesDFd.iloc[i][j] = 0
advisoriesDFd = advisoriesDFd.add_prefix('advisories_')
print(advisoriesDFd)

#Add feature being the number of advisories an app has
advisoriesCount = catega.groupby(level=0).count()
advisoriesCount.plot()
advisoriesNumberDF = []
for nbr in range (len(advisoriesCount)):
    if (advisoriesCount[nbr] < 4):
        advisoriesNumberDF.append('<4')
    elif (advisoriesCount[nbr] < 9):
        advisoriesNumberDF.append('[4-9[')
    else:
        advisoriesNumberDF.append('[9-')

advisoriesNumberDF = pd.DataFrame(advisoriesNumberDF, columns =␣
 ↪['advisoriesNumber'])
advisoriesNumberDF['advisoriesNumber'] = pd.
 ↪Categorical(advisoriesNumberDF['advisoriesNumber'])
advisoriesNumberDFd = pd.get_dummies(advisoriesNumberDF['advisoriesNumber'])
advisoriesNumberDFd = advisoriesNumberDFd.add_prefix('advisoriesNumber_')
print(advisoriesNumberDFd)

#artistName -> change type to categorical
dfCleaned['artistName'] = pd.Categorical(dfCleaned['artistName'])

#artistViewUrl -> change type to categorical
dfCleaned['artistViewUrl'] = pd.Categorical(dfCleaned['artistViewUrl'])
#Changed to a binary variable where we check whether seller published his url or␣
 ↪nah
artistViewUrlDF = dfCleaned['artistViewUrl']
artistViewUrlDF = artistViewUrlDF.cat.add_categories([0, 1])
artistViewUrlDFd = pd.DataFrame(index=dfCleaned.index, columns=['artistViewUrl'])
artistViewUrlDFb = (pd.isnull(np.array(artistViewUrlDF)))
```

```
for i in range (len(artistViewUrlDFb)):
    if (artistViewUrlDFb[i]):
        artistViewUrlDFd.iloc[i]['artistViewUrl'] = 0
    else:
        artistViewUrlDFd.iloc[i]['artistViewUrl'] = 1
print(artistViewUrlDFd)

#contentAdvisoryRating -> change type to categorical
dfCleaned['contentAdvisoryRating'] = pd.
 ↪Categorical(dfCleaned['contentAdvisoryRating'])

#currentVersionReleaseDate -> change type to datetime
dfCleaned['currentVersionReleaseDate'] = pd.
 ↪to_datetime(dfCleaned['currentVersionReleaseDate'])
#Create dummies with years -> INITIAL COLUMN 2 DROP IN FINAL DUMMY DATA
currentVersionReleaseDateCat = dfCleaned['currentVersionReleaseDate']
currentVersionReleaseDateCat = currentVersionReleaseDateCat.dt.year
currentVersionReleaseDateDF = []
for dat in range (len(currentVersionReleaseDateCat)):
    if (currentVersionReleaseDateCat[dat] < 2015):
        currentVersionReleaseDateDF.append('<2015')
    elif (currentVersionReleaseDateCat[dat] < 2020):
        currentVersionReleaseDateDF.append('[2015-2020[')
    else:
        currentVersionReleaseDateDF.append('2020')
currentVersionReleaseDateDF = pd.Categorical(currentVersionReleaseDateDF)
currentVersionReleaseDateDFd = pd.get_dummies(currentVersionReleaseDateDF)
currentVersionReleaseDateDFd = currentVersionReleaseDateDFd.
 ↪add_prefix('currentVersionReleaseDate_')
print(currentVersionReleaseDateDFd)

#description -> change type to categorical -> 2 drop for now bc they are all␣
 ↪different and all present
dfCleaned['description'] = pd.Categorical(dfCleaned['description'])

#features -> replace '[]' with nan & change type to categorical
dfCleaned['features'] = dfCleaned['features'].replace('[]', np.nan)
dfCleaned['features'] = pd.Categorical(dfCleaned['features'])

#formattedPrice -> change type to categorical
dfCleaned['formattedPrice'] = pd.Categorical(dfCleaned['formattedPrice'])

#genreIds -> Drop column bc brings the same information as genres -> will create␣
 ↪unnecessary correlation in models
dfCleaned = dfCleaned.drop(['genreIds'], axis=1)
```

```
#artistId -> Drop column bc brings the same information as artistName -> will␣
 ↪create unnecessary correlation in models
dfCleaned = dfCleaned.drop(['artistId'], axis=1)

#userRatingCountForCurrentVersion -> Drop column bc brings the same information␣
 ↪as userRatingCount -> will create unnecessary correlation in models
dfCleaned = dfCleaned.drop(['userRatingCountForCurrentVersion'], axis=1)

#primaryGenreId -> Drop column bc brings the same information as␣
 ↪primaryGenreName -> will create unnecessary correlation in models
dfCleaned = dfCleaned.drop(['primaryGenreId'], axis=1)

#averageUserRatingForCurrentVersion -> Drop column bc brings the same␣
 ↪information as averageUserRating -> will create unnecessary correlation in␣
 ↪models
dfCleaned = dfCleaned.drop(['averageUserRatingForCurrentVersion'], axis=1)

#genres -> change type to categorical
dfCleaned['genres'] = pd.Categorical(dfCleaned['genres'])
#Create dummies with categories -> 2 check again -> INITIAL COLUMN 2 DROP IN␣
 ↪FINAL DUMMY DATA
gen = dfCleaned['genres']
genClean = [x.strip('[]').split(',') for x in gen]

categg = []
for a in genClean:
    categg.append(a)

categg = pd.DataFrame(categg).stack()
categg = categg.str.strip()
genresCateg = categg.unique()
genresCateg = np.sort(genresCateg)

genresDFd = pd.DataFrame(index=dfCleaned.index, columns=genresCateg)
for i in range (len(gen)):
    for j in range(len(genresCateg)):
        if (genresCateg[j] in gen[i]):
            genresDFd.iloc[i][j] = 1
        else:
            genresDFd.iloc[i][j] = 0
genresDFd = genresDFd.add_prefix('genres_')
print(genresDFd)

#ipadScreenshotUrls -> change type to categorical
dfCleaned['ipadScreenshotUrls'] = pd.Categorical(dfCleaned['ipadScreenshotUrls'])
#Create dummies with categories -> 2 check again -> INITIAL COLUMN 2 DROP IN␣
 ↪FINAL DUMMY DATA
```

```
ipadScreenshotUrlsDF = dfCleaned['ipadScreenshotUrls']
ipadScreenshotUrlsDF = ipadScreenshotUrlsDF.cat.add_categories([0, 1])
ipadScreenshotUrlsDF[ipadScreenshotUrlsDF == '[]'] = 0
ipadScreenshotUrlsDF[ipadScreenshotUrlsDF != 0] = 1
ipadScreenshotUrlsDFd = pd.DataFrame(ipadScreenshotUrlsDF)
ipadScreenshotUrlsDFd = ipadScreenshotUrlsDFd.add_prefix('ipadScreenshotUrls_')
print(ipadScreenshotUrlsDFd)

#isGameCenterEnabled -> change type to categorical
dfCleaned['isGameCenterEnabled'] = pd.
 ↪Categorical(dfCleaned['isGameCenterEnabled'])

#languageCodesISO2A -> change type to categorical
dfCleaned['languageCodesISO2A'] = pd.Categorical(dfCleaned['languageCodesISO2A'])
#Create dummies with categories -> 2 check again -> INITIAL COLUMN 2 DROP IN␣
 ↪FINAL DUMMY DATA
lan = dfCleaned['languageCodesISO2A']
lanClean = [x.strip('[]').split(',') for x in lan]

categ = []
for a in lanClean:
    categ.append(a)

categ = pd.DataFrame(categ).stack()
categ = categ.str.strip()
languageCodesISO2ACateg = categ.unique()
languageCodesISO2ACateg = np.sort(languageCodesISO2ACateg)

languageCodesISO2ADFdd = pd.DataFrame(index=dfCleaned.index,␣
 ↪columns=languageCodesISO2ACateg)
for i in range (len(lan)):
    for j in range(len(languageCodesISO2ACateg)):
        if (languageCodesISO2ACateg[j] in lan[i]):
            languageCodesISO2ADFdd.iloc[i][j] = 1
        else:
            languageCodesISO2ADFdd.iloc[i][j] = 0
languageCodesISO2ADFdd = languageCodesISO2ADFdd.drop([''], axis=1)
languageCodesISO2ADFdd = languageCodesISO2ADFdd.add_prefix('languageCodesISO2A_')
print(languageCodesISO2ADFdd)

#Number of languages
NbrlanguageCodesISO2A = categ.groupby(level=0).count()
NbrlanguageCodesISO2A.plot()
NbrlanguageCodesISO2ADF = []
for nbrl in range (len(NbrlanguageCodesISO2A)):
    if (NbrlanguageCodesISO2A[nbrl] < 5):
        NbrlanguageCodesISO2ADF.append('<5')
```

```
    elif (NbrlanguageCodesISO2A[nbrl] < 20):
        NbrlanguageCodesISO2ADF.append('[5-20[')
    elif (NbrlanguageCodesISO2A[nbrl] < 45):
        NbrlanguageCodesISO2ADF.append('[20-45[')
    else:
        NbrlanguageCodesISO2ADF.append('[45-')
NbrlanguageCodesISO2ADF = pd.DataFrame(NbrlanguageCodesISO2ADF, columns =␣
 ↪['NbrlanguageCodesISO2A'])
NbrlanguageCodesISO2ADF['NbrlanguageCodesISO2A'] = pd.
 ↪Categorical(NbrlanguageCodesISO2ADF['NbrlanguageCodesISO2A'])
NbrlanguageCodesISO2ADFd = pd.
 ↪get_dummies(NbrlanguageCodesISO2ADF['NbrlanguageCodesISO2A'])
NbrlanguageCodesISO2ADFd = NbrlanguageCodesISO2ADFd.
 ↪add_prefix('NbrlanguageCodesISO2A')
print(NbrlanguageCodesISO2ADFd)

#minimumOsVersion -> change type to categorical
dfCleaned['minimumOsVersion'] = pd.Categorical(dfCleaned['minimumOsVersion'])
#Create dummies with categories -> INITIAL COLUMN 2 DROP OR REPLACE IN FINAL␣
 ↪DUMMY DATA??? Not sure
minimumOsVersionDF = dfCleaned['minimumOsVersion'].str.rsplit('.').str[0]
minimumOsVersionDFd = pd.get_dummies(minimumOsVersionDF)
minimumOsVersionDFd = minimumOsVersionDFd.add_prefix('minimumOsVersion_')
print(minimumOsVersionDFd)

#primaryGenreName -> change type to categorical
dfCleaned['primaryGenreName'] = pd.Categorical(dfCleaned['primaryGenreName'])

#releaseDate -> change type to datetime
dfCleaned['releaseDate'] = pd.to_datetime(dfCleaned['releaseDate'])
#Create dummies with months -> INITIAL COLUMN 2 DROP IN FINAL DUMMY DATA
releaseDateCat = dfCleaned['releaseDate']
releaseDateCat = releaseDateCat.dt.month
releaseDateDF = pd.Categorical(releaseDateCat)
releaseDateDFd = pd.get_dummies(releaseDateDF)
releaseDateDFd = releaseDateDFd.add_prefix('releaseDate_')
print(releaseDateDFd)



#releaseNotes -> change type to categorical
dfCleaned['releaseNotes'] = pd.Categorical(dfCleaned['releaseNotes'])
#Create dummies with categories -> 2 check again -> INITIAL COLUMN 2 DROP IN␣
 ↪FINAL DUMMY DATA
#Changed to a binary variable where we check whether seller published notes or␣
 ↪nah
```

```
releaseNotesDF = dfCleaned['releaseNotes']
releaseNotesDF = releaseNotesDF.cat.add_categories([0, 1])
releaseNotesDF = releaseNotesDF.fillna(0)
releaseNotesDF[releaseNotesDF != 0] = 1
releaseNotesDFd = pd.DataFrame(releaseNotesDF)
releaseNotesDFd = releaseNotesDFd.add_prefix('releaseNotes_')
print(releaseNotesDFd)

#screenshotUrls -> change type to categorical
dfCleaned['screenshotUrls'] = pd.Categorical(dfCleaned['screenshotUrls'])
#Create dummies with categories -> 2 check again -> INITIAL COLUMN 2 DROP IN␣
 ↪FINAL DUMMY DATA
#Changed to a binary variable where we check whether seller published the␣
 ↪screenshot urls or nah
screenshotUrlsDF = dfCleaned['screenshotUrls']
screenshotUrlsDF = screenshotUrlsDF.cat.add_categories([0, 1])
screenshotUrlsDF[screenshotUrlsDF == '[]'] = 0
screenshotUrlsDF[screenshotUrlsDF != 0] = 1
screenshotUrlsDFd = pd.DataFrame(screenshotUrlsDF)
screenshotUrlsDFd = screenshotUrlsDFd.add_prefix('screenshotUrls_')
print(screenshotUrlsDFd)

#sellerName -> change type to categorical
dfCleaned['sellerName'] = pd.Categorical(dfCleaned['sellerName'])

#sellerUrl -> change type to categorical
dfCleaned['sellerUrl'] = pd.Categorical(dfCleaned['sellerUrl'])
#Create dummies with categories -> 2 check again -> INITIAL COLUMN 2 DROP IN␣
 ↪FINAL DUMMY DATA
#Changed to a binary variable where we check whether seller published his url or␣
 ↪nah
sellerUrlDF = dfCleaned['sellerUrl']
sellerUrlDF = sellerUrlDF.cat.add_categories([0, 1])
sellerUrlDF = sellerUrlDF.fillna(0)
sellerUrlDF[sellerUrlDF != 0] = 1
sellerUrlDFd = pd.DataFrame(sellerUrlDF)
sellerUrlDFd = sellerUrlDFd.add_prefix('sellerUrl_')
print(sellerUrlDFd)

#supportedDevices -> change type to categorical
dfCleaned['supportedDevices'] = pd.Categorical(dfCleaned['supportedDevices'])
supDev = dfCleaned['supportedDevices']
supDevClean = [x.strip('[]').split(',') for x in supDev]

categsd = []
for a in supDevClean:
    categsd.append(a)
```

```
categsd = pd.DataFrame(categsd).stack()
categsd = categsd.str.strip()
supDevCateg = categsd.unique()
supDevCateg = np.sort(supDevCateg)

supportedDevicesDFd = pd.DataFrame(index=dfCleaned.index, columns=supDevCateg)
for i in range (len(supDev)):
    for j in range(len(supDevCateg)):
        if (supDevCateg[j] in supDev[i]):
            supportedDevicesDFd.iloc[i][j] = 1
        else:
            supportedDevicesDFd.iloc[i][j] = 0
supportedDevicesDFd = supportedDevicesDFd.add_prefix('supportedDevices_')
print(supportedDevicesDFd)

#trackCensoredName -> Drop column bc brings the same information as trackName ->␣
 ↪will create unnecessary correlation in models
z = ((dfCleaned['trackCensoredName']) == (dfCleaned['trackName']))
print(z.value_counts())
dfCleaned = dfCleaned.drop(['trackCensoredName'], axis=1)

#trackContentRating -> change type to categorical
dfCleaned['trackContentRating'] = pd.Categorical(dfCleaned['trackContentRating'])

#trackName -> change type to categorical
dfCleaned['trackName'] = pd.Categorical(dfCleaned['trackName'])

#version -> change type to categorical
dfCleaned['version'] = pd.Categorical(dfCleaned['version'])
#Create dummies with categories -> INITIAL COLUMN 2 DROP OR REPLACE IN FINAL␣
 ↪DUMMY DATA??? Not sure
versionDF = dfCleaned['version'].str.split(".").str[0]
versionDF = versionDF.replace('2020 a tour in BTS soul map', '2020')
versionDF = versionDF.replace('2019 Winter Bear BTS come back', '2019')
versionDF = versionDF.str.replace('22OCT2019-V', '')
versionDF = versionDF.str.replace(r'[a-z]+', '')
versionDF = versionDF.str.replace(r'[A-Z]+', '')
versionDF = versionDF.str.replace('_1', '')
versionDF = versionDF.replace('', '1')
versionDF = versionDF.replace('1,1,7', '1')
versionDF = versionDF.replace('1,0120', '1')
versionDF = versionDF.str.strip()
versionDFd = pd.get_dummies(versionDF)
versionDFd = versionDFd.add_prefix('version_')
print(versionDFd)
```

```
#price -> change type to categorical -> 2drop bc brings same info as␣
 ↪formattedPrice
dfCleaned['price'] = pd.Categorical(dfCleaned['price'])
dfCleaned = dfCleaned.drop(['price'], axis=1)

# Change fileSizeBytes into classes of small, medium and large
fileSizeVal = dfCleaned['fileSizeBytes']
fileSizeVal = np.array(fileSizeVal)
fileSizeVal = fileSizeVal.astype(int)
print(min(dfCleaned['fileSizeBytes'].unique()))
print(max(dfCleaned['fileSizeBytes'].unique()))

#Create Class for Apps -> small < 30 MB, medium >= 30 MB & < 50 MB, big >= 50 MB
# 30MB = 30000000 B; 50MB = 50000000B
fileSizeClassMB = []
print(fileSizeVal[5])
for val in range (len(fileSizeVal)):
    if (fileSizeVal[val] == -9223372036854775808):
        fileSizeClassMB.append(np.nan)
    elif (fileSizeVal[val] < 30000000):
        fileSizeClassMB.append('Small')
    elif (fileSizeVal[val] < 50000000):
        fileSizeClassMB.append('Medium')
    elif (fileSizeVal[val] >= 50000000):
        fileSizeClassMB.append('Big')
fileSizeClassMB = pd.DataFrame(fileSizeClassMB, columns =['fileSizeClassMB'])
fileSizeClassMBd = pd.get_dummies(fileSizeClassMB['fileSizeClassMB'],␣
 ↪dummy_na=True)
fileSizeClassMBd = fileSizeClassMBd.add_prefix('fileSizeClassMB_')
print(fileSizeClassMBd)

#index -> drop bc no clue what it represents and no further information
dfCleaned = dfCleaned.drop(['index'], axis=1)

#userRatingCount -> change type to categorical
dfCleaned['userRatingCount'] = pd.Categorical(dfCleaned['userRatingCount'])
#Check the amount of ratings to create classes
#<10 = 5K ; <50 = 9.5K ; <100 = 11k ; <1000 = 16k ; >50000 = 1K
ctr = 0
for cnt in range (len(dfCleaned['userRatingCount'])):
    if (dfCleaned.iloc[cnt]['userRatingCount'] < 10):
        ctr += 1
ctr

#Based on that, create 7 classes
userRatingCountDF = []
userRatingCountVal = dfCleaned['userRatingCount']
```

```
for vall in range (len(userRatingCountVal)):
    if (userRatingCountVal[vall] < 10):
        userRatingCountDF.append('[1-10[')
    elif (userRatingCountVal[vall] < 50):
        userRatingCountDF.append('[10-50[')
    elif (userRatingCountVal[vall] < 100):
        userRatingCountDF.append('[50-100[')
    elif (userRatingCountVal[vall] < 1000):
        userRatingCountDF.append('[100-1000[')
    elif (userRatingCountVal[vall] < 10000):
        userRatingCountDF.append('[1000-10000[')
    elif (userRatingCountVal[vall] < 50000):
        userRatingCountDF.append('[10000-50000[')
    else:
        userRatingCountDF.append('[50000-')

userRatingCountDF = pd.DataFrame(userRatingCountDF, columns =['userRatingCount'])
userRatingCountDFd = pd.get_dummies(userRatingCountDF['userRatingCount'])
userRatingCountDFd = userRatingCountDFd.add_prefix('userRatingCount_')
print(userRatingCountDFd)
```

[ ]:
```
#OUTPUT

#Change'averageUserRating' into categories to predict CLASS not values
dfCleaned['averageUserRating'] = pd.Categorical(dfCleaned['averageUserRating'])
#At some point, necessity to create .5 classes bc major difference btw app of␣
 ↪rating of 3 and 3.9
averageUserRatingDF = []
averageUserRatingVal = dfCleaned['averageUserRating']
for val in range (len(averageUserRatingVal)):
    if (averageUserRatingVal[val] == 0):
        averageUserRatingDF.append(np.nan)
    elif (averageUserRatingVal[val] < 1):
        averageUserRatingDF.append(0)
    elif (averageUserRatingVal[val] < 2):
        averageUserRatingDF.append(1)
    elif (averageUserRatingVal[val] < 2.5):
        averageUserRatingDF.append(2)
    elif (averageUserRatingVal[val] < 3):
        averageUserRatingDF.append(2.5)
    elif (averageUserRatingVal[val] < 3.5):
        averageUserRatingDF.append(3)
    elif (averageUserRatingVal[val] < 4):
        averageUserRatingDF.append(3.5)
    elif (averageUserRatingVal[val] < 4.5):
        averageUserRatingDF.append(4)
    elif (averageUserRatingVal[val] < 5):
```

```
        averageUserRatingDF.append(4.5)
    elif (averageUserRatingVal[val] == 5):
        averageUserRatingDF.append(5)

averageUserRatingDF = pd.DataFrame(averageUserRatingDF, columns␣
 ↪=['averageUserRating'])
averageUserRatingDF['averageUserRating'] = pd.
 ↪Categorical(averageUserRatingDF['averageUserRating'])
print(averageUserRatingDF)
averageUserRatingDFd = pd.get_dummies(averageUserRatingDF['averageUserRating'])
(averageUserRatingDFd)
```

1.3 Mise en place du dataset final

[ ]:
```
print(dfCleaned.info())

dfCleanedFinal = dfCleaned.drop(['artistName', 'sellerName', 'primaryGenreName',␣
 ↪'averageUserRating','currentVersionReleaseDate', 'trackName', 'genres',␣
 ↪'ipadScreenshotUrls', 'languageCodesISO2A', 'minimumOsVersion',␣
 ↪'supportedDevices', 'releaseDate', 'releaseNotes', 'screenshotUrls',␣
 ↪'sellerUrl', 'version', 'description', 'artistViewUrl', 'advisories',␣
 ↪'fileSizeBytes', 'userRatingCount'], axis=1)

print(dfCleanedFinal.info())

DummyDF1 = pd.get_dummies(dfCleanedFinal, dummy_na=True)
DummyDF1
```

[ ]:
```
#Do not integrate versionDFd, currentVersionReleaseDateDFd, UserRatingsCount␣
 ↪because impossible to make recommandations about that
DummyDF2 = pd.concat([genresDFd, supportedDevicesDFd, sellerUrlDFd,␣
 ↪screenshotUrlsDFd, releaseNotesDFd, releaseDateDFd, minimumOsVersionDFd,␣
 ↪ipadScreenshotUrlsDFd, artistViewUrlDFd, advisoriesDFd, advisoriesNumberDFd,␣
 ↪fileSizeClassMBd, NbrlanguageCodesISO2ADFd], axis = 1)
DummyDF2
```

[ ]:
```
DummyDF = pd.concat([DummyDF1, DummyDF2], axis = 1)
outputRatings = averageUserRatingDF['averageUserRating'].astype(str)
featNames = list(DummyDF.columns)
targNames = list(averageUserRatingDFd.columns)
DummyDF
```

[ ]:
```
#Save dummy dataset
DummyDF.to_csv(r'/Users/patienceumuhoza/Documents/Mémoire/DonnéesAppStore/
 ↪dummyDatasetThesis.csv', index = False, header=True)
```

1.4 Algorithmes de classification

[ ]:
```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score




%matplotlib inline
#Affichage de la figure
```

[ ]:
```
#Create plot with Apps per Categories
GenreVal = dfCleaned.groupby('primaryGenreName').size()
GenreValList = list(GenreVal.index)
plt.figure(figsize=(20,12))
plt.barh(GenreValList, GenreVal)
plt.title('Nombre d\'applications par categorie')
plt.savefig('plotAppCat.png', dpi=300, bbox_inches='tight')

plt.show()
```

[ ]:
```
#Create plot with Ratings count Categories

ratVal = (userRatingCountDF['userRatingCount'].value_counts())
print(ratVal)
ratNames = list(ratVal.index)
plt.barh(ratNames, ratVal)
plt.title('Quantité de ratings par classe')
plt.savefig('plotRatCount.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#Create plot with Ratings count Categories

avRatVal = (averageUserRatingDF['averageUserRating'].value_counts())
print(avRatVal)
avRatNames = list(avRatVal.index)
```

```
plt.bar(avRatNames, avRatVal, color = 'red', tick_label = avRatNames, edgecolor␣
 ↪= 'black', width = 0.4)
plt.title('Nombre d\'évaluations par classe de ratings')
plt.savefig('plotAVGRatCount.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#Set Data for models with the genres
primaryGenreNameDFd = pd.get_dummies(dfCleaned['primaryGenreName'])
DummyGenDF = pd.concat([DummyDF, primaryGenreNameDFd], axis = 1)
X = np.array(DummyGenDF)
y = outputRatings
topFeatures = pd.DataFrame()
print(DummyGenDF)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[ ]:
```
yz=[]

for i in range(0,266):
    pca = PCA(n_components=i+1)
    pca.fit(X)
    yz.append(sum(pca.explained_variance_ratio_))

plt.figure()
x=range(0,266)
plt.plot(x,yz,'-r',c='#F23E44')
plt.title("Variance expliquée par composantes principales sur les données␣
 ↪brutes")
plt.xlabel("Nombre de composantes principales")
plt.ylabel("Total de variance expliquée")
plt.savefig('explVar.png', dpi=300, bbox_inches='tight')
plt.show()

print("La variance expliquée par les composantes principales est de", yz[49])
```

[ ]:
```
#Select first 50 features bc they explain more than 80% of the variance
pca = PCA(n_components=50)
data_acp = pca.fit_transform(X)
print(data_acp)
data_acp.shape
```

[ ]:
```
#Find optimal K for kmeans model
kClusters = range(1, 10)
inertias = []
for k in kClusters:
    model = KMeans(n_clusters=k)
```

```
    model.fit(X_embedded)
    inertias.append(model.inertia_)

plt.plot(kClusters, inertias, '-o', color='black')
plt.xlabel('Nombre de clusters')
plt.ylabel('Inertie')
plt.title('Evolution de l\'inertie par rapport au nombre de clusters')
plt.xticks(kClusters)
plt.savefig('inertClust.png', dpi=300, bbox_inches='tight')

plt.show()
```

[ ]:
```
perplValues = [5, 15, 30, 45, 50]
kClusters = [3, 4, 5]

averageUserRatingDF.iloc[closestInst, :]
for i in range (len(perplValues)):
    X_embedded = TSNE(n_components=2, perplexity=perplValues[i]).
 ↪fit_transform(data_acp)
    for k in range (len(kClusters)):
        kmeanModel = KMeans(n_clusters=kClusters[k])
        kmeanModel.fit(X_embedded)
        kmeanPred = kmeanModel.predict(X_embedded)
        centers = kmeanModel.cluster_centers_
        plt.scatter(X_embedded[:, 0], X_embedded[:, 1], s=1, c=kmeanPred)
        plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
        plt.title("Clustering par k-Means avec k=" + str(kClusters[k]) +" et␣
 ↪perplexité=" + str(perplValues[i]))
        plt.show()
        closestInst, _ = pairwise_distances_argmin_min(centers, X_embedded)
        closestInst
        print(averageUserRatingDF.iloc[closestInst, :])
```

[ ]:
```
X_embedded = TSNE(n_components=2, perplexity=45).fit_transform(data_acp)
kmeanModel = KMeans(n_clusters=4)
kmeanModel.fit(X_embedded)
kmeanPred = kmeanModel.predict(X_embedded)
centers = kmeanModel.cluster_centers_
plt.figsize=(50,25)
plt.figure(figsize=(8, 6))
plt.scatter(X_embedded[:, 0], X_embedded[:, 1], s=1, c=kmeanPred)
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=75, alpha=0.5)
plt.title('Clustering par k-Means avec 4 clusters et une perplexité de 45')
plt.savefig('ClustFin.png', dpi=300, bbox_inches='tight')
#for i, txt in enumerate(y):
#    ax[1].annotate(txt, (X_embedded[i, 0], X_embedded[i, 1]))
```

1.4.1 Modèle général

[ ]:
```
ValTot = pd.DataFrame(y_train)
ValTotFreq = ValTot['averageUserRating'].value_counts()
ValTotNames = list(ValTotFreq.index)
plt.bar(ValTotNames, ValTotFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriTot.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#Set up gridsearch RF
rf = RandomForestClassifier(random_state = 11)
parameters = {'criterion': ['gini','entropy'], 'n_estimators': [5, 10, 15, 25,␣
 ↪35, 50, 75, 100], 'max_features': ['sqrt', 'auto'], 'bootstrap': [True, False]}
```

[ ]:
```
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train, y_train)
prediction_forest_test = grid.predict(X_test)
print(grid.best_params_)
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 100)
clf = clf.fit(X_train, y_train)
prediction = clf.predict(X_train)
predictiont = clf.predict(X_test)
#confusion matrix
print(confusion_matrix(y_test, predictiont))
#Metric f1_score
print('f1 score training set:', f1_score(y_train, prediction,␣
 ↪average='weighted'))
print('f1 score testing set:',f1_score(y_test, predictiont, average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
#topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[46 13 10 46 20 81 88 31] [19 8 16 26 10 62 67 22] [13 15 19 24 35 78 106 18] [23 13 17 51 47 204 181 36] [16 19 21 65 80 275 309 33] [23 26 19 71 87 597 748 72] [24 12 14 60 71 530 1551 53] [20 10 10 30 42 158 197 76]] f1 score training set: 0.9794573971881658 f1 score testing set: 0.3240814498813137

1.4.2 Social Networking

[ ]:
```
#'Social Networking' Dataset
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFSocNet = grouped.get_group('Social Networking')
averageUserRatingDFdSocNet = outputRatings[DummyDFSocNet.index]

#Setting training and testing sets
X = np.array(DummyDFSocNet)
y = np.array(averageUserRatingDFdSocNet)

X_train_SocNet, X_test_SocNet, y_train_SocNet, y_test_SocNet =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
DummyDFSocNet
```

[ ]:
```
ValSocNet = pd.DataFrame(y_train_SocNet, columns = ['valeurs'])
ValSocNetFreq = ValSocNet['valeurs'].value_counts()
ValSocNetNames = list(ValSocNetFreq.index)
plt.bar(ValSocNetNames, ValSocNetFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriSocNet.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_SocNet, y_train_SocNet)
prediction_forest_test = grid.predict(X_test_SocNet)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 75}
```

[ ]:
```
#RF
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_SocNet, y_train_SocNet)
prediction = clf.predict(X_train_SocNet)
predictiont = clf.predict(X_test_SocNet)
print('Accuracy training set:', accuracy_score(y_train_SocNet, prediction,␣
 ↪normalize=True))
print('Accuracy testing set:', accuracy_score(y_test_SocNet, predictiont,␣
 ↪normalize=True))
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_SocNet, y_train_SocNet)
prediction = clf.predict(X_train_SocNet)
predictiont = clf.predict(X_test_SocNet)
```

```
#confusion matrix
print(confusion_matrix(y_test_SocNet, predictiont))
#Metric f1_score
print('f1 score training set:', f1_score(y_train_SocNet, prediction,␣
 ↪average='weighted'))
print('f1 score testing set:',f1_score(y_test_SocNet, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

Accuracy training set: 0.9949558638083228 Accuracy testing set: 0.31671554252199413 [[4 2 0 4 1 6 1 0] [0 2 0 1 0 3 3 0] [0 1 1 1 0 3 5 0] [0 1 1 4 1 16 8 1] [1 0 1 3 6 22 12 4] [2 3 2 3 6 52 23 6] [2 0 0 4 2 48 34 2] [2 0 2 1 0 14 9 5]] f1 score training set: 0.9949544842510978 f1 score testing set: 0.29513359728741584

1.4.3 Medical

[ ]:
```
#'Medical' Dataset
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFMedical = grouped.get_group(GenreList[1])
averageUserRatingDFdMedical = outputRatings[DummyDFMedical.index]

#Setting training and testing sets
X = np.array(DummyDFMedical)
y = np.array(averageUserRatingDFdMedical)

X_train_Medical, X_test_Medical, y_train_Medical, y_test_Medical =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValMedical = pd.DataFrame(y_train_Medical, columns = ['valeurs'])
ValMedicalFreq = ValMedical['valeurs'].value_counts()
ValMedicalNames = list(ValMedicalFreq.index)
plt.bar(ValMedicalNames, ValMedicalFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriMedical.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Medical, y_train_Medical)
```

```
prediction_forest_test = grid.predict(X_test_Medical)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 100}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 100)
clf = clf.fit(X_train_Medical, y_train_Medical)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Medical)
predictiont = clf.predict(X_test_Medical)
#confusion matrix
print(confusion_matrix(y_test_Medical, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Medical, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
#f1 score training set: 0.358974358974359
```

[[0 0 0 0 2 1 1] [0 0 0 1 0 1 0] [0 0 0 0 2 0 0] [0 0 0 0 0 2 0] [0 0 0 0 0 3 0] [0 0 1 0 1 14 0] [0 0 0 0 1 2 0]] f1 score testing set: 0.358974358974359

[ ]:
```
#'Entertainment' Dataset
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFEntertainment = grouped.get_group(GenreList[2])
averageUserRatingDFdEntertainment = outputRatings[DummyDFEntertainment.index]

#Setting training and testing sets
X = np.array(DummyDFEntertainment)
y = np.array(averageUserRatingDFdEntertainment)

X_train_Entertainment, X_test_Entertainment, y_train_Entertainment,␣
 ↪y_test_Entertainment = train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValEntertainment = pd.DataFrame(y_train_Entertainment, columns = ['valeurs'])
ValEntertainmentFreq = ValEntertainment['valeurs'].value_counts()
ValEntertainmentNames = list(ValEntertainmentFreq.index)
plt.bar(ValEntertainmentNames, ValEntertainmentFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
```

```
plt.savefig('plotDistriEntertainment.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Entertainment, y_train_Entertainment)
prediction_forest_test = grid.predict(X_test_Entertainment)
print(grid.best_params_)
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 100)
clf = clf.fit(X_train_Entertainment, y_train_Entertainment)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Entertainment)
predictiont = clf.predict(X_test_Entertainment)
#confusion matrix
print(confusion_matrix(y_test_Entertainment, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Entertainment, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
#0.2829165263896243
```

[[2 3 1 4 3 6 8 1] [0 3 3 1 2 13 6 0] [2 0 2 1 7 10 10 0] [1 0 3 1 8 14 24 2] [2 1 4 4 10 23 26 1] [2 2 0 11 10 53 67 0] [2 4 5 3 7 35 105 0] [4 1 3 3 3 8 8 1]] f1 score testing set: 0.2829165263896243

[ ]:
```
#'Lifestyle'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFLifestyle = grouped.get_group(GenreList[3])
averageUserRatingDFdLifestyle = outputRatings[DummyDFLifestyle.index]

#Setting training and testing sets
X = np.array(DummyDFLifestyle)
y = np.array(averageUserRatingDFdLifestyle)

X_train_Lifestyle, X_test_Lifestyle, y_train_Lifestyle, y_test_Lifestyle =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValLifestyle = pd.DataFrame(y_train_Lifestyle, columns = ['valeurs'])
ValLifestyleFreq = ValLifestyle['valeurs'].value_counts()
ValLifestyleNames = list(ValLifestyleFreq.index)
plt.bar(ValLifestyleNames, ValLifestyleFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriLifestyle.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Lifestyle, y_train_Lifestyle)
prediction_forest_test = grid.predict(X_test_Lifestyle)
print(grid.best_params_)
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'entropy', max_features = 'sqrt', n_estimators = 25)
clf = clf.fit(X_train_Lifestyle, y_train_Lifestyle)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Lifestyle)
predictiont = clf.predict(X_test_Lifestyle)
#confusion matrix
print(confusion_matrix(y_test_Lifestyle, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Lifestyle, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
#0.26345135137013737
```

[[4 1 1 1 1 3 9 0] [0 0 2 0 2 3 5 0] [0 0 0 0 1 4 12 0] [0 0 2 5 1 3 4 1] [2 0 1 3 2 6 11 0] [2 1 3 3 5 16 35 7] [2 1 1 1 4 20 52 8] [0 0 0 2 2 6 13 2]] f1 score testing set: 0.26345135137013737

[ ]:
```
#'Navigation'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFNavigation = grouped.get_group(GenreList[4])
averageUserRatingDFdNavigation = outputRatings[DummyDFNavigation.index]

#Setting training and testing sets
X = np.array(DummyDFNavigation)
y = np.array(averageUserRatingDFdNavigation)
```

```
X_train_Navigation, X_test_Navigation, y_train_Navigation, y_test_Navigation =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValNavigation = pd.DataFrame(y_train_Navigation, columns = ['valeurs'])
ValNavigationFreq = ValNavigation['valeurs'].value_counts()
ValNavigationNames = list(ValNavigationFreq.index)
plt.bar(ValNavigationNames, ValNavigationFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriNavigation.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Navigation, y_train_Navigation)
prediction_forest_test = grid.predict(X_test_Navigation)
print(grid.best_params_)
#{'bootstrap': False, 'criterion': 'entropy', 'max_features': 'sqrt',␣
 ↪'n_estimators': 50}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'entropy', max_features = 'sqrt', n_estimators = 50)
clf = clf.fit(X_train_Navigation, y_train_Navigation)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Navigation)
predictiont = clf.predict(X_test_Navigation)
#confusion matrix
print(confusion_matrix(y_test_Navigation, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Navigation, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
#0.155103995621237
```

[[0 0 0 0 1 2 2 1] [1 0 0 1 3 1 0 0] [1 0 0 1 3 1 1 0] [0 3 0 3 2 3 0 2] [0 1 0 0 0 4 4 1] [0 0 0 0 1 2 4 4] [0 0 0 0 1 2 5 3] [0 1 0 0 0 3 2 2]] f1 score testing set: 0.155103995621237

[ ]:
```
#'Games'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
```

```
DummyDFGames = grouped.get_group('Games')
averageUserRatingDFdGames = outputRatings[DummyDFGames.index]

#Setting training and testing sets
X = np.array(DummyDFGames)
y = np.array(averageUserRatingDFdGames)

X_train_Games, X_test_Games, y_train_Games, y_test_Games = train_test_split(X,␣
 ↪y, test_size=0.3, random_state=42)
```

[ ]:
```
ValGames = pd.DataFrame(y_train_Games, columns = ['valeurs'])
ValGamesFreq = ValGames['valeurs'].value_counts()
ValGamesNames = list(ValGamesFreq.index)
plt.bar(ValGamesNames, ValGamesFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriGames.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Games, y_train_Games)
prediction_forest_test = grid.predict(X_test_Games)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'entropy', 'max_features': 'sqrt',␣
 ↪'n_estimators': 100}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'entropy', max_features = 'sqrt', n_estimators = 100)
clf = clf.fit(X_train_Games, y_train_Games)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Games)
predictiont = clf.predict(X_test_Games)
#confusion matrix
print(confusion_matrix(y_test_Games, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Games, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[3 3 4 9 6 25 8 11] [4 5 5 6 9 29 8 5] [6 2 4 9 10 37 19 4] [4 7 4 22 28 85 56 14] [1 6 4 22 34 101

93 10] [5 10 7 33 49 276 315 20] [5 3 1 18 30 205 559 14] [7 2 2 17 27 54 54 19]] f1 score testing set: 0.344782138243747

[ ]:
```
#'Reference'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFReference = grouped.get_group(GenreList[6])
averageUserRatingDFdReference = outputRatings[DummyDFReference.index]

#Setting training and testing sets
X = np.array(DummyDFReference)
y = np.array(averageUserRatingDFdReference)

X_train_Reference, X_test_Reference, y_train_Reference, y_test_Reference =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValReference = pd.DataFrame(y_train_Reference, columns = ['valeurs'])
ValReferenceFreq = ValReference['valeurs'].value_counts()
ValReferenceNames = list(ValReferenceFreq.index)
plt.bar(ValReferenceNames, ValReferenceFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriReference.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Reference, y_train_Reference)
prediction_forest_test = grid.predict(X_test_Reference)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 75}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_Reference, y_train_Reference)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Reference)
predictiont = clf.predict(X_test_Reference)
#confusion matrix
print(confusion_matrix(y_test_Reference, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Reference, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
```

```
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[1 0 0 1 2 4 2 3] [0 0 0 0 1 1 2 0] [1 0 1 0 2 2 2 0] [0 0 0 0 0 1 2 3] [0 0 0 0 4 2 9 2] [1 0 1 0 4 4 16 0] [0 0 1 0 1 12 41 2] [2 0 0 0 2 3 5 1]] f1 score testing set: 0.3212013930151185

[ ]:
```
#'Photo & Video'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFPV = grouped.get_group(GenreList[7])
averageUserRatingDFdPV = outputRatings[DummyDFPV.index]

#Setting training and testing sets
X = np.array(DummyDFPV)
y = np.array(averageUserRatingDFdPV)

X_train_PV, X_test_PV, y_train_PV, y_test_PV = train_test_split(X, y,␣
 ↪test_size=0.3, random_state=42)
```

[ ]:
```
ValPV = pd.DataFrame(y_train_PV, columns = ['valeurs'])
ValPVFreq = ValPV['valeurs'].value_counts()
ValPVNames = list(ValPVFreq.index)
plt.bar(ValPVNames, ValPVFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriPV.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_PV, y_train_PV)
prediction_forest_test = grid.predict(X_test_PV)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'entropy', 'max_features': 'sqrt',␣
 ↪'n_estimators': 75}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'entropy', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_PV, y_train_PV)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_PV)
predictiont = clf.predict(X_test_PV)
#confusion matrix
print(confusion_matrix(y_test_PV, predictiont))
#Metric f1_score
```

```
print('f1 score testing set:', f1_score(y_test_PV, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[5 2 0 4 2 10 5 3] [0 3 0 2 1 3 4 1] [1 2 0 1 1 7 8 0] [3 0 1 2 4 16 10 3] [2 0 0 4 2 22 23 3] [8 0 1 5 9 42 67 5] [2 0 2 1 6 50 114 7] [2 1 1 2 2 8 10 2]] f1 score testing set: 0.3039862409569209

[ ]:
```
# 'Book'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFBook = grouped.get_group(GenreList[8])
averageUserRatingDFdBook = outputRatings[DummyDFBook.index]

#Setting training and testing sets
X = np.array(DummyDFBook)
y = np.array(averageUserRatingDFdBook)

X_train_Book, X_test_Book, y_train_Book, y_test_Book = train_test_split(X, y,␣
 ↪test_size=0.3, random_state=42)
```

[ ]:
```
ValBook = pd.DataFrame(y_train_Book, columns = ['valeurs'])
ValBookFreq = ValBook['valeurs'].value_counts()
ValBookNames = list(ValBookFreq.index)
plt.bar(ValBookNames, ValBookFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriBook.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Book, y_train_Book)
prediction_forest_test = grid.predict(X_test_Book)
print(grid.best_params_)
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'entropy', max_features = 'sqrt', n_estimators = 5)
clf = clf.fit(X_train_Book, y_train_Book)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Book)
predictiont = clf.predict(X_test_Book)
```

```
#confusion matrix
print(confusion_matrix(y_test_Book, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Book, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[1 0 0 1 0 1 2 0] [0 0 0 1 0 0 1 0] [0 0 0 0 0 1 3 0] [0 0 0 0 0 3 0 0] [0 0 1 0 0 2 2 0] [1 0 0 2 2 5 8 1] [0 0 1 1 3 11 20 2] [1 0 0 0 0 4 4 0]] f1 score testing set: 0.292560167879861

[ ]:
```
#'Music'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFMusic = grouped.get_group(GenreList[9])
averageUserRatingDFdMusic = outputRatings[DummyDFMusic.index]

#Setting training and testing sets
X = np.array(DummyDFMusic)
y = np.array(averageUserRatingDFdMusic)

X_train_Music, X_test_Music, y_train_Music, y_test_Music = train_test_split(X,␣
 ↪y, test_size=0.3, random_state=42)
```

[ ]:
```
ValMusic = pd.DataFrame(y_train_Music, columns = ['valeurs'])
ValMusicFreq = ValMusic['valeurs'].value_counts()
ValMusicNames = list(ValMusicFreq.index)
plt.bar(ValMusicNames, ValMusicFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriMusic.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Music, y_train_Music)
prediction_forest_test = grid.predict(X_test_Music)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 50}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 50)
clf = clf.fit(X_train_Music, y_train_Music)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Music)
predictiont = clf.predict(X_test_Music)
#confusion matrix
print(confusion_matrix(y_test_Music, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Music, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[2 2 0 3 2 9 4 2] [0 0 1 0 2 7 4 1] [0 1 3 1 2 4 4 0] [0 4 3 4 5 10 7 2] [0 2 3 1 7 14 10 4] [1 5 3 6 12 33 38 5] [4 1 3 5 8 21 46 2] [3 1 1 1 1 6 4 10]] f1 score testing set: 0.2881151096205504

[ ]:
```
# 'Productivity'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFProductivity = grouped.get_group(GenreList[10])
averageUserRatingDFdProductivity = outputRatings[DummyDFProductivity.index]

#Setting training and testing sets
X = np.array(DummyDFProductivity)
y = np.array(averageUserRatingDFdProductivity)

X_train_Productivity, X_test_Productivity, y_train_Productivity,␣
 ↪y_test_Productivity = train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValProductivity = pd.DataFrame(y_train_Productivity, columns = ['valeurs'])
ValProductivityFreq = ValProductivity['valeurs'].value_counts()
ValProductivityNames = list(ValProductivityFreq.index)
plt.bar(ValProductivityNames, ValProductivityFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriProductivity.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Productivity, y_train_Productivity)
```

```
prediction_forest_test = grid.predict(X_test_Productivity)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 75}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_Productivity, y_train_Productivity)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Productivity)
predictiont = clf.predict(X_test_Productivity)
#confusion matrix
print(confusion_matrix(y_test_Productivity, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Productivity, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[1 1 1 0 0 5 5 0] [0 0 0 0 0 2 2 4 0] [0 0 0 0 0 1 3 5 0] [1 0 3 2 1 6 14 0] [1 2 3 0 2 6 14 0] [1 0 1 1 2 11 40 1] [2 0 1 2 3 14 60 3] [1 1 1 0 0 6 10 0]] f1 score testing set: 0.2519706079485332

[ ]:
```
#'Shopping'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFShopping = grouped.get_group(GenreList[11])
averageUserRatingDFdShopping = outputRatings[DummyDFShopping.index]

#Setting training and testing sets
X = np.array(DummyDFShopping)
y = np.array(averageUserRatingDFdShopping)

X_train_Shopping, X_test_Shopping, y_train_Shopping, y_test_Shopping =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValShopping = pd.DataFrame(y_train_Shopping, columns = ['valeurs'])
ValShoppingFreq = ValShopping['valeurs'].value_counts()
ValShoppingNames = list(ValShoppingFreq.index)
plt.bar(ValShoppingNames, ValShoppingFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriShopping.png', dpi=300, bbox_inches='tight')
```

```
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Shopping, y_train_Shopping)
prediction_forest_test = grid.predict(X_test_Shopping)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 35}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 35)
clf = clf.fit(X_train_Shopping, y_train_Shopping)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Shopping)
predictiont = clf.predict(X_test_Shopping)
#confusion matrix
print(confusion_matrix(y_test_Shopping, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Shopping, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[1 0 1 0 0 1 0 1] [0 0 0 0 0 0 3 0] [0 0 0 1 0 0 2 0] [0 0 0 2 0 3 5 0] [0 1 0 1 1 4 9 0] [1 0 2 1 1 4 8 1] [0 1 0 0 3 8 36 1] [0 0 0 0 0 1 7 0]] f1 score testing set: 0.3501187030598795

[ ]:
```
# 'Utilities'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFUtilities = grouped.get_group(GenreList[12])
averageUserRatingDFdUtilities = outputRatings[DummyDFUtilities.index]

#Setting training and testing sets
X = np.array(DummyDFUtilities)
y = np.array(averageUserRatingDFdUtilities)

X_train_Utilities, X_test_Utilities, y_train_Utilities, y_test_Utilities =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValUtilities = pd.DataFrame(y_train_Utilities, columns = ['valeurs'])
ValUtilitiesFreq = ValUtilities['valeurs'].value_counts()
ValUtilitiesNames = list(ValUtilitiesFreq.index)
plt.bar(ValUtilitiesNames, ValUtilitiesFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriUtilities.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Utilities, y_train_Utilities)
prediction_forest_test = grid.predict(X_test_Utilities)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'entropy', 'max_features': 'sqrt',␣
 ↪'n_estimators': 100}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'entropy', max_features = 'sqrt', n_estimators = 100)
clf = clf.fit(X_train_Utilities, y_train_Utilities)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Utilities)
predictiont = clf.predict(X_test_Utilities)
#confusion matrix
print(confusion_matrix(y_test_Utilities, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Utilities, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[3 2 2 1 1 7 5 1] [1 4 1 2 2 5 3 1] [3 3 4 1 2 2 7 1] [2 1 4 1 5 12 10 0] [5 1 3 1 3 12 19 0] [4 1 2 5 6 36 43 1] [2 1 3 5 5 50 60 1] [0 1 0 1 1 14 6 1]] f1 score testing set: 0.2633392536884258

[ ]:
```
# 'Education',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFEducation = grouped.get_group(GenreList[13])
averageUserRatingDFdEducation = outputRatings[DummyDFEducation.index]

#Setting training and testing sets
X = np.array(DummyDFEducation)
```

```
y = np.array(averageUserRatingDFdEducation)

X_train_Education, X_test_Education, y_train_Education, y_test_Education =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValEducation = pd.DataFrame(y_train_Education, columns = ['valeurs'])
ValEducationFreq = ValEducation['valeurs'].value_counts()
ValEducationNames = list(ValEducationFreq.index)
plt.bar(ValEducationNames, ValEducationFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriEducation.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Education, y_train_Education)
prediction_forest_test = grid.predict(X_test_Education)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 35}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 35)
clf = clf.fit(X_train_Education, y_train_Education)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Education)
predictiont = clf.predict(X_test_Education)
#confusion matrix
print(confusion_matrix(y_test_Education, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Education, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[1 1 0 1 1 6 4 3] [0 1 0 3 0 3 2 1] [0 2 0 0 0 3 4 2] [0 0 1 1 1 8 3 3] [2 0 0 0 2 12 20 4] [1 0 0 1 1 29 18 1] [0 0 1 0 3 20 33 7] [3 3 0 4 2 7 15 3]] f1 score testing set: 0.23435800301810727

[ ]:
```
# 'Health & Fitness',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
```

```
DummyDFHF = grouped.get_group(GenreList[14])
averageUserRatingDFdHF = outputRatings[DummyDFHF.index]

#Setting training and testing sets
X = np.array(DummyDFHF)
y = np.array(averageUserRatingDFdHF)

X_train_HF, X_test_HF, y_train_HF, y_test_HF = train_test_split(X, y,␣
 ↪test_size=0.3, random_state=42)
```

[ ]:
```
ValHF = pd.DataFrame(y_train_HF, columns = ['valeurs'])
ValHFFreq = ValHF['valeurs'].value_counts()
ValHFNames = list(ValHFFreq.index)
plt.bar(ValHFNames, ValHFFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriHF.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_HF, y_train_HF)
prediction_forest_test = grid.predict(X_test_HF)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 75}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_HF, y_train_HF)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_HF)
predictiont = clf.predict(X_test_HF)
#confusion matrix
print(confusion_matrix(y_test_HF, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_HF, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[0 0 0 1 0 3 3 0] [0 1 0 1 0 1 8 0] [0 0 0 0 0 0 4 0] [0 0 0 0 1 3 13 0] [0 0 0 2 1 4 8 0] [1 0 1 1 3 9 28 0] [

0 0 0 0 3 8 84 1] [0 0 0 1 0 0 10 0]] f1 score testing set: 0.3800742774620122

[ ]:
```
# 'Sports',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFSports = grouped.get_group(GenreList[15])
averageUserRatingDFdSports = outputRatings[DummyDFSports.index]

#Setting training and testing sets
X = np.array(DummyDFSports)
y = np.array(averageUserRatingDFdSports)

X_train_Sports, X_test_Sports, y_train_Sports, y_test_Sports =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValSports = pd.DataFrame(y_train_Sports, columns = ['valeurs'])
ValSportsFreq = ValSports['valeurs'].value_counts()
ValSportsNames = list(ValSportsFreq.index)
plt.bar(ValSportsNames, ValSportsFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriSports.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Sports, y_train_Sports)
prediction_forest_test = grid.predict(X_test_Sports)
print(grid.best_params_)
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_Sports, y_train_Sports)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Sports)
predictiont = clf.predict(X_test_Sports)
#confusion matrix
print(confusion_matrix(y_test_Sports, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Sports, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
```

```
importanceDF[:10]
```

[[1 0 0 2 0 0 1 0] [1 0 0 0 0 0 3 0] [0 0 0 0 0 1 5 0] [0 0 0 1 0 1 1 0] [0 0 0 6 1 0 5 0] [3 1 1 0 0 5 13 2] [0 1 1 3 1 5 24 0] [0 0 0 2 2 1 4 2]] f1 score testing set: 0.3071564922865232

[ ]:
```
# 'News',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFNews = grouped.get_group(GenreList[16])
averageUserRatingDFdNews = outputRatings[DummyDFNews.index]

#Setting training and testing sets
X = np.array(DummyDFNews)
y = np.array(averageUserRatingDFdNews)

X_train_News, X_test_News, y_train_News, y_test_News = train_test_split(X, y,␣
 ↪test_size=0.3, random_state=42)
```

[ ]:
```
ValNews = pd.DataFrame(y_train_News, columns = ['valeurs'])
ValNewsFreq = ValNews['valeurs'].value_counts()
ValNewsNames = list(ValNewsFreq.index)
plt.bar(ValNewsNames, ValNewsFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriNews.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_News, y_train_News)
prediction_forest_test = grid.predict(X_test_News)
print(grid.best_params_)
#{'bootstrap': False, 'criterion': 'entropy', 'max_features': 'sqrt',␣
 ↪'n_estimators': 25}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = False, criterion =␣
 ↪'entropy', max_features = 'sqrt', n_estimators = 25)
clf = clf.fit(X_train_News, y_train_News)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_News)
predictiont = clf.predict(X_test_News)
#confusion matrix
print(confusion_matrix(y_test_News, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_News, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
```

```
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[0 2 1 0 1 0 0 0] [0 0 3 0 0 1 2 0] [0 2 0 2 1 0 0 2] [0 2 1 1 1 3 4 0] [0 0 1 0 0 2 2 1] [0 0 0 1 3 6 7 1] [1 1 1 2 2 6 31 1] [1 0 1 0 1 2 4 0]] f1 score testing set: 0.3400557468437449

[ ]:
```
# 'Business',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFBusiness = grouped.get_group(GenreList[17])
averageUserRatingDFdBusiness = outputRatings[DummyDFBusiness.index]

#Setting training and testing sets
X = np.array(DummyDFBusiness)
y = np.array(averageUserRatingDFdBusiness)

X_train_Business, X_test_Business, y_train_Business, y_test_Business =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValBusiness = pd.DataFrame(y_train_Business, columns = ['valeurs'])
ValBusinessFreq = ValBusiness['valeurs'].value_counts()
ValBusinessNames = list(ValBusinessFreq.index)
plt.bar(ValBusinessNames, ValBusinessFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriBusiness.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Business, y_train_Business)
prediction_forest_test = grid.predict(X_test_Business)
print(grid.best_params_)
#{'bootstrap': False, 'criterion': 'entropy', 'max_features': 'sqrt',␣
 ↪'n_estimators': 35}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = False, criterion =␣
 ↪'entropy', max_features = 'sqrt', n_estimators = 35)
clf = clf.fit(X_train_Business, y_train_Business)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Business)
predictiont = clf.predict(X_test_Business)
#confusion matrix
print(confusion_matrix(y_test_Business, predictiont))
```

```
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Business, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[3 0 0 0 2 0 3 1] [0 0 0 0 0 0 2 3 0] [1 0 1 0 0 1 1 2] [5 1 0 0 3 4 3 3] [0 1 2 1 0 5 5 4] [5 1 0 0 1 10 10 3] [4 1 2 1 4 8 23 3] [4 1 0 0 2 2 7 3]] f1 score testing set: 0.23987341433634024

[ ]:
```
# 'Weather',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFWeather = grouped.get_group(GenreList[18])
averageUserRatingDFdWeather = outputRatings[DummyDFWeather.index]

#Setting training and testing sets
X = np.array(DummyDFWeather)
y = np.array(averageUserRatingDFdWeather)

X_train_Weather, X_test_Weather, y_train_Weather, y_test_Weather =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValWeather = pd.DataFrame(y_train_Weather, columns = ['valeurs'])
ValWeatherFreq = ValWeather['valeurs'].value_counts()
ValWeatherNames = list(ValWeatherFreq.index)
plt.bar(ValWeatherNames, ValWeatherFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriWeather.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Weather, y_train_Weather)
prediction_forest_test = grid.predict(X_test_Weather)
print(grid.best_params_)
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 100)
clf = clf.fit(X_train_Weather, y_train_Weather)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Weather)
```

```
predictiont = clf.predict(X_test_Weather)
#confusion matrix
print(confusion_matrix(y_test_Weather, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Weather, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[0 0 0 0 0 0 1 0 0] [0 0 0 1 0 0 2 0] [0 0 1 1 2 1 1 0] [0 0 2 1 1 1 0 0] [0 0 0 4 2 1 10 0] [0 0 0 2 1 3 8 0] [1 0 0 0 0 4 29 0] [0 0 0 2 0 2 0 0]] f1 score testing set: 0.3750256745210161

[ ]:
```
# 'Stickers',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFStickers = grouped.get_group(GenreList[19])
averageUserRatingDFdStickers = outputRatings[DummyDFStickers.index]

#Setting training and testing sets
X = np.array(DummyDFStickers)
y = np.array(averageUserRatingDFdStickers)

X_train_Stickers, X_test_Stickers, y_train_Stickers, y_test_Stickers =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValStickers = pd.DataFrame(y_train_Stickers, columns = ['valeurs'])
ValStickersFreq = ValStickers['valeurs'].value_counts()
ValStickersNames = list(ValStickersFreq.index)
plt.bar(ValStickersNames, ValStickersFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriStickers.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Stickers, y_train_Stickers)
prediction_forest_test = grid.predict(X_test_Stickers)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 75}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_Stickers, y_train_Stickers)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Stickers)
predictiont = clf.predict(X_test_Stickers)
#confusion matrix
print(confusion_matrix(y_test_Stickers, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Stickers, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[2 0 0 0 0 0 0 0] [1 0 0 0 0 0 0 1] [1 0 0 0 0 0 0 0] [0 0 0 0 0 0 2 3] [0 0 0 1 0 0 0 0] [0 0 0 1 0 0 0 0] [0 0 0 0 1 0 2 0] [0 0 0 0 0 0 0 1]] f1 score testing set: 0.21130952380952384

[ ]:
```
# 'Finance',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFFinance = grouped.get_group(GenreList[20])
averageUserRatingDFdFinance = outputRatings[DummyDFFinance.index]

#Setting training and testing sets
X = np.array(DummyDFFinance)
y = np.array(averageUserRatingDFdFinance)

X_train_Finance, X_test_Finance, y_train_Finance, y_test_Finance =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValFinance = pd.DataFrame(y_train_Finance, columns = ['valeurs'])
ValFinanceFreq = ValFinance['valeurs'].value_counts()
ValFinanceNames = list(ValFinanceFreq.index)
plt.bar(ValFinanceNames, ValFinanceFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriFinance.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Finance, y_train_Finance)
```

```
prediction_forest_test = grid.predict(X_test_Finance)
print(grid.best_params_)
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
↪'gini', max_features = 'sqrt', n_estimators = 15)
clf = clf.fit(X_train_Finance, y_train_Finance)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Finance)
predictiont = clf.predict(X_test_Finance)
#confusion matrix
print(confusion_matrix(y_test_Finance, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Finance, predictiont,␣
↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[0 0 0 0 0 0 1 0 0] [0 0 0 0 0 0 0 0 1 0] [0 0 0 0 0 1 1 0 0] [1 0 0 1 0 0 3 0] [0 0 1 0 0 5 6 0] [0 2 1 2 1 10 21 2] [0 1 0 1 1 11 33 1] [0 0 1 0 0 2 6 1]] f1 score testing set: 0.3448431185837978

[ ]:
```
# 'Food & Drink',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFFD = grouped.get_group(GenreList[21])
averageUserRatingDFdFD = outputRatings[DummyDFFD.index]

#Setting training and testing sets
X = np.array(DummyDFFD)
y = np.array(averageUserRatingDFdFD)

X_train_FD, X_test_FD, y_train_FD, y_test_FD = train_test_split(X, y,␣
↪test_size=0.3, random_state=42)
```

[ ]:
```
ValFD = pd.DataFrame(y_train_FD, columns = ['valeurs'])
ValFDFreq = ValFD['valeurs'].value_counts()
ValFDNames = list(ValFDFreq.index)
plt.bar(ValFDNames, ValFDFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriFD.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_FD, y_train_FD)
prediction_forest_test = grid.predict(X_test_FD)
print(grid.best_params_)
#{'bootstrap': True, 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators':
 ↪ 75}
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_FD, y_train_FD)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_FD)
predictiont = clf.predict(X_test_FD)
#confusion matrix
print(confusion_matrix(y_test_FD, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_FD, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[0 1 0 0 0 0 1 1] [0 1 0 0 0 0 0 1] [0 2 0 0 0 0 0 0] [0 1 0 0 0 1 3 4] [0 0 0 0 0 2 2 3] [0 2 1 0 0 0 3 2] [0 0 1 1 1 2 12 11] [1 2 1 3 0 1 2 11]] f1 score testing set: 0.2761957813428401

[ ]:
```
# 'Graphics & Design',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFGD = grouped.get_group(GenreList[22])
averageUserRatingDFdGD = outputRatings[DummyDFGD.index]

#Setting training and testing sets
X = np.array(DummyDFGD)
y = np.array(averageUserRatingDFdGD)

X_train_GD, X_test_GD, y_train_GD, y_test_GD = train_test_split(X, y,␣
 ↪test_size=0.3, random_state=42)
```

[ ]:
```
ValGD = pd.DataFrame(y_train_GD, columns = ['valeurs'])
ValGDFreq = ValGD['valeurs'].value_counts()
ValGDNames = list(ValGDFreq.index)
```

```
plt.bar(ValGDNames, ValGDFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriGD.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_GD, y_train_GD)
prediction_forest_test = grid.predict(X_test_GD)
print(grid.best_params_)
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 100)
clf = clf.fit(X_train_GD, y_train_GD)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_GD)
predictiont = clf.predict(X_test_GD)
#confusion matrix
print(confusion_matrix(y_test_GD, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_GD, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[0 0 0 1] [0 0 0 1] [0 0 0 7] [0 0 2 9]] f1 score testing set: 0.34137931034482755

[ ]:
```
# 'Travel',
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFTravel = grouped.get_group(GenreList[23])
averageUserRatingDFdTravel = outputRatings[DummyDFTravel.index]

#Setting training and testing sets
X = np.array(DummyDFTravel)
y = np.array(averageUserRatingDFdTravel)

X_train_Travel, X_test_Travel, y_train_Travel, y_test_Travel =␣
 ↪train_test_split(X, y, test_size=0.3, random_state=42)
```

[ ]:
```
ValTravel = pd.DataFrame(y_train_Travel, columns = ['valeurs'])
ValTravelFreq = ValTravel['valeurs'].value_counts()
ValTravelNames = list(ValTravelFreq.index)
plt.bar(ValTravelNames, ValTravelFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriTravel.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```
#RF
grid = GridSearchCV(rf, param_grid=parameters, cv=None)
grid.fit(X_train_Travel, y_train_Travel)
prediction_forest_test = grid.predict(X_test_Travel)
print(grid.best_params_)
```

[ ]:
```
clf = RandomForestClassifier(random_state = 11, bootstrap = True, criterion =␣
 ↪'gini', max_features = 'sqrt', n_estimators = 75)
clf = clf.fit(X_train_Travel, y_train_Travel)
#tree.plot_tree(clf)
prediction = clf.predict(X_train_Travel)
predictiont = clf.predict(X_test_Travel)
#confusion matrix
print(confusion_matrix(y_test_Travel, predictiont))
#Metric f1_score
print('f1 score testing set:', f1_score(y_test_Travel, predictiont,␣
 ↪average='weighted'))
#Features importance
importance_ = list(zip(featNames, clf.feature_importances_))
importanceDF = pd.DataFrame(importance_, columns = ['Features', 'Importance'])
importanceDF = importanceDF[importanceDF['Importance'] > 0]
importanceDF = importanceDF.sort_values(by='Importance', ascending=False)
topFeatures = pd.concat([topFeatures, importanceDF.iloc[:10, 0]])
print(importanceDF)
importanceDF[:10]
```

[[0 0 1 0 0 0 1 1] [0 0 0 1 0 0 3 0] [0 0 1 0 0 0 4 0] [0 0 0 0 0 6 4 0] [1 0 1 0 0 2 3 1] [2 0 0 0 1 4 10 2] [0 0 1 1 0 6 24 0] [0 0 0 0 0 0 3 2 0]] f1 score testing set: 0.2722922698546122

[ ]:
```
# 'Magazines & Newspapers'
GenreList = list(dfCleaned['primaryGenreName'].unique())
grouped = DummyDF.groupby(dfCleaned.primaryGenreName)
DummyDFMN = grouped.get_group(GenreList[24])
averageUserRatingDFdMN = outputRatings[DummyDFMN.index]

#Setting training and testing sets
X = np.array(DummyDFMN)
y = np.array(averageUserRatingDFdMN)
```

```
X_train_MN, X_test_MN, y_train_MN, y_test_MN = train_test_split(X, y,␣
↪test_size=0.3, random_state=42)
```

[ ]:
```
ValMN = pd.DataFrame(y_train_MN, columns = ['valeurs'])
ValMNFreq = ValMN['valeurs'].value_counts()
ValMNNames = list(ValMNFreq.index)
plt.bar(ValMNNames, ValMNFreq)
plt.title('Distribution des ratings de l\'ensemble d\'entraînement')
plt.savefig('plotDistriMN.png', dpi=300, bbox_inches='tight')
plt.show()
```

[ ]:
```

```